11 (рекурсивные алгоритмы)

Тема: Рекурсия. Рекурсивные процедуры и функции

Что нужно знать:

- для того, чтобы задать рекурсивную функцию, нужно определить
  1) условие окончания рекурсии, то есть значения параметров функции, для которых значение функции известно или вычисляется без рекурсивных вызовов;
  2) рекуррентную формулу (или формулы), с помощью которых значение функции для заданных значений параметров вычисляется через значение (или значения) функции для других значений параметров (то есть, с помощью рекурсивных вызовов)
- задачи, в которых требуется найти значение заданной рекурсивной функции при известных значениях параметров можно решать с помощью ручных вычислений, используя электронные таблицы или с помощью своей программы; последние два способа обычно более эффективны
- функцию

  $F(n) = 1$ при $n \le 1$

  $F(n) = n + 1 + F(n–1)$, при $n > 1$

  можно реализовать следующим образом в виде функций на языках программирования:

  Python:

```
def F( n ):
  if n <= 1:
    return 1
  else:
    return n + 1 + F(n-1)
```

  Паскаль:

```
function F( n: integer ): integer;
begin
  if n <= 1 then
    Result := 1;
  else
    Result := n + 1 + F(n-1);
end;
```

  C++:

```
int F( int n )
{
  if( n <= 1 )
    return 1;
  else
    return n + 1 + F(n-1);
}
```

Пример задания:

P-03. Определите наименьшее значение n, при котором сумма чисел, которые будут выведены при вызове $F(n)$, будет больше 500000. Запишите в ответе сначала найденное значение n, а затем через пробел – соответствующую сумму выведенных чисел.

Python	Паскаль	C++

def F(n):
  print(2*n)
  if n > 1:
	procedure F
	  (n: integer);
	begin
		void F(int n)
		{
		  cout << 2*n << endl;

```
    print(n-5)
    F(n-1)
    F(n-2)
```

```
  writeln(2*n);
  if n > 1 then begin
    writeln(n-5);
    F(n-1);
    F(n-2);
  end;
end;
```

```
  if( n > 1 ) {
    cout << n-5 << endl;
    F(n-1);
    F(n-2);
    }
}
```

Для выполнения задания можно также написать программу или воспользоваться редактором электронных таблиц.

Решение (с помощью счётчика):

1) первое, что может прийти в голову – вызывать приведённую процедуру при разных значениях параметра и увеличивать это значение до тех пор, пока сумма выведенных чисел не превысит заданное значение 500000; это тупиковый подход, поскольку чисел очень много и сложение займет очень много времени при низкой вероятности правильного ответа
2) можно попробовать изменить программу так, чтобы сумма выводимых чисел считалась автоматически: добавим в программу глобальную переменную **s** и будем увеличивать её при выводе каждого числа на значение этого числа; при этом для ускорения (значительного!) работы программы сразу закомментируем вывод чисел на экран:

```
def F( n ):
  global s    # если не объявить s глобальной – ошибка!
  # print(2*n)
  s += 2*n
  if n > 1:
    # print(n-5)
    s += n - 5
    F(n-1)
    F(n-2)
```

3) дальше можно написать такую программу и запускать её при различных значениях переменной **n**:

```
n = 15
s = 0
F(n)
print( n, s )
```

4) увеличивая каждый раз значение n на 1, мы в конце концов найдём первое (минимальное) значение n, при котором сумма чисел, которые будут выведены при вызове $F(n)$, будет больше 500000 – это $F(24) = 531864$
5) можно оформить поиск нужного значения n в виде цикла, например, так:

```
n = 0
while True:
  n += 1        # первое значение n = 1
  s = 0         # нужно обнулять сумму перед каждым вызовом
  F(n)          # подсчитали сумму
  if s > 500000: break; # если нашли, выход из цикла
print( n, s )   # вывод результата
```

6) решение на языке Паскаль (строки с выводом чисел закомментированы):

```
var s, n: integer;
procedure F( n: integer );
begin
  // writeln(2*n);
```

```
    s := s + 2*n;
    if n > 1 then begin
      // writeln(n-5);
      s := s + n - 5;
      F(n-1);
      F(n-2);
    end;
  end;
  begin
    n := 0;
    repeat
      n := n + 1;
      s := 0;
      F(n);
    until s > 500000;
    writeln( n, ' ', s);
  end.
```

7) решение на языке C++ (строки с выводом чисел закомментированы):

```
  #include <iostream>
  using namespace std;
  int s;
  void F( int n )
  {
    // cout << 2*n << endl;
    s += 2*n;
    if( n > 1 ) {
      // cout << n-5 << endl;
      s += n - 5;
      F(n-1);
      F(n-2);
      }
  }
  int main()
  {
    int n = 0;
    do {
      n += 1;
      s = 0;
      F(n);
      }
    while( s <= 500000 );
    cout << n << " " << s;
  }
```

8) в этой задаче процедура вызывает сама себя дважды, такая «ветвистость» может значительно увеличить время вычислений; если вы видите, что программа работает очень долго (результата не дождаться), нужно применить другой подход – искать решение через рекуррентную формулу (см. следующее решение) и применять метод динамического программирования, сохраняя в массиве все промежуточные результаты (см. подробности в решении задачи P-01)

9) Ответ: 24 531864.

Решение (с помощью формулы):

1) возможен другой приём: построить рекурсивную функцию $f(n)$, которая определяет сумму выведенных чисел; после этого можно использовать любой из методов вычислений, рассмотренных ниже для задачи P-01 – считать вручную, использовать электронную таблицу или написать свою собственную программу
2) итак, согласно условию, при $n \le 1$ выводится только число $2n$, то есть

   $f(n) = 2n$ при $n \le 1$

   это условие окончания рекурсии
3) основная часть определения рекурсивной функции – рекуррентная формула для остальных случаев; при $n > 1$ процедура печатает число $2n$ сразу при входе в процедуру, затем – ещё число $n–5$ в теле условного оператора, а затем дважды вызывает сама себя с разными значениями параметра, так что

   $f(n) = 2n + n – 5 + f(n–1) + f(n–2)$ при $n > 1$
4) теперь эту функцию можно записать, например, в виде программного кода:

```
def f( n ):
  if n <= 1:
    return 2*n
  else:
    return 2*n + n – 5 + f(n-1) + f(n-2)
```

5) запишем основную программу – алгоритм поиска нужного значения n:

```
n = 0
while True:
  n += 1
  s = f(n)
  if s > 500000: break;
print( n, s )
```

6) решение на языке Паскаль:

```
var s, n: integer;
function f( n: integer ): integer;
begin
  if n <= 1 then
    Result := 2*n
  else
    Result := 2*n - 5 + f(n-1) + f(n-2);
end;
begin
  n := 0;
  repeat
    n := n + 1;
    s := f(n);
  until s > 500000;
  writeln( n, ' ', s);
end.
```

7) решение на языке C++:

```
#include <iostream>
using namespace std;
int f( int n )
{
  if( n <= 1 )
    return 2*n;
```

```
    else
      return 2*n + n - 5 + f(n-1) + f(n-2);
}
int main()
{
  int n = 0, s;
  do {
    n += 1;
    s = f(n);
    }
  while( s <= 500000 );

  cout << n << " " << s;
}
```

8) заметим, что когда мы получили математическую форму записи функции (см. пп. 2 и 3), для решения задачи можно использовать любой из методов, описанных в разборе задачи P-01 ниже (например, ручной счёт или электронные таблицы)
9) недостаток такого подхода в том, что он *косвенный*, то есть требует преобразований; поэтому есть неплохой шанс ошибиться при выводе формулы и поэтому получить неверный результат
10) если вы видите, что программа работает очень долго (результата не дождаться), нужно применять метод динамического программирования, сохраняя в массиве все промежуточные результаты (см. подробности в решении задачи P-01)
11) Ответ: 24 531864.

Ещё пример задания:

P-02. Определите, сколько символов * выведет эта процедура при вызове $F(22)$:

Python	Паскаль	C++

def F(n):
  print('*')
  if n >= 1:
  print('*')
  F(n-1)
  F(n-2)
  F(n-3)
	procedure F(n: integer);
	begin
	  write('*');
	  if n >= 1 then begin
	  write('*');
	  F(n-1);
	  F(n-2);
	  F(n-3);
	  end;
	end;
		void F(int n)
		{
		  cout << '*';
		  if(n >= 1) {
		  cout << '*';
		  F(n-1);
		  F(n-2);
		  F(n-3);
		  }
		}

Для выполнения задания можно также написать программу или воспользоваться редактором электронных таблиц.

Решение (с помощью счётчика):

1) может показаться, что эта задача легче, чем задача P-01, разобранная ниже: ведь нам уже дана реализация функции на языке программирования, остаётся только запустить её и посмотреть, что она выведет
2) первое впечатление очень обманчиво; при вызове $F(22)$ программа выводит огромное количество звёздочек (**больше миллиона!**), подсчитать их вручную не представляется возможным

3) можно попробовать изменить программу так, чтобы выводимые звёздочки считались автоматически: добавим в программу глобальную переменную-счётчик **count** и будем увеличивать его при выводе каждой звёздочки:

```
count = 0
def F( n ):
  global count # если не объявить count глобальной – ошибка!
  print('*')
  count += 1
  if n >= 1:
    print('*')
    count += 1
    F(n-1)
    F(n-2)
    F(n-3)

F(22)
print(count)
```

4) запуск программы с вызовов $F(22)$ показывает, что очень много времени занимает вывод звёздочек на экран, мы рискуем не дождаться ответа

5) поэтому убираем из программы операторы вывода в теле функции (например, их можно временно отключить с помощью комментариев):

```
count = 0
def F( n ):
  global count # если не объявить count глобальной – ошибка!
  # print('*')
  count += 1
  if n >= 1:
    # print('*')
    count += 1
    F(n-1)
    F(n-2)
    F(n-3)

F(22)
print(count)
```

6) запустив такую программу, быстро получаем ответ: 1957585

7) решение на языке Паскаль (строки с выводом звёздочек закомментированы):

```
var count: integer;
procedure F( n: integer );
begin
  // write('*');
  count := count + 1;
  if n >= 1 then begin
    // write('*');
    count := count + 1;
    F(n-1);
    F(n-2);
    F(n-3);
  end;
end;
```

```
begin
  count := 0;
  F(22);
  writeln(count);
end.
```

8) решение на языке C++ (строки с выводом звёздочек закомментированы):

```
#include <iostream>
using namespace std;
int count = 0;
void F( int n )
{
  // cout << '*';
  count ++;
  if( n >= 1 ) {
    // cout << '*';
    count ++;
    F(n-1);
    F(n-2);
    F(n-3);
    }
}
int main()
{
  F(22);
  cout << count;
}
```

9) в этой задаче процедура вызывает сама себя трижды, такая «ветвистость» может значительно увеличить время вычислений; если вы видите, что программа работает очень долго (результата не дождаться), нужно применить другой подход – искать решение через рекуррентную формулу (см. следующее решение) и применять метод динамического программирования, сохраняя в массиве все промежуточные результаты (см. подробности в решении задачи P-01)
10) Ответ: 1957585.

Решение (с помощью формулы):

1) возможен другой приём: построить рекурсивную функцию $f(n)$, которая определяет количество выведенных звёздочек; после этого можно использовать любой из методов вычислений, рассмотренных ниже для задачи P-01 – считать вручную, использовать электронную таблицу или написать свою собственную программу
2) итак, согласно условию, при $n < 1$ выводится только одна звёздочка, то есть

   $f(n) = 1$ при $n < 1$

   это условие окончания рекурсии
3) основная часть определения рекурсивной функции – рекуррентная формула для остальных случаев; при $n \ge 1$ процедура печатает одну звёздочку сразу при входе в процедуру, затем – ещё одну в теле условного оператора, а затем трижды вызывает сама себя с разными значениями параметра, так что

   $f(n) = 1 + 1 + f(n–1) + f(n–2) + f(n–3)$ при $n \ge 1$
4) теперь эту функцию можно записать, например, в виде программного кода:

```
def f( n ):
  if n < 1:
```

```
    return 1
  else:
    return 2 + f(n-1) + f(n-2) + f(n-3)
```

вызвать и вывести полученный ответ:

```
print( f(22) )
```

5) решение на языке Паскаль:

```
function f( n: integer ): integer;
begin
  if n < 1 then
    Result := 1
  else
    Result := 2 + f(n-1) + f(n-2) + f(n-3);
end;
begin
  writeln( f(22) );
end.
```

6) решение на языке C++:

```
#include <iostream>
using namespace std;
int f( int n )
{
  if( n < 1 )
    return 1;
  else
    return 2 + f(n-1) + f(n-2) + f(n-3);
}
int main()
{
  cout << f(22);
}
```

7) заметим, что когда мы получили математическую форму записи функции (см. пп. 2 и 3), для вычисления $f(22)$ можно использовать любой из методов, описанных в разборе задачи P-01 ниже (например, ручной счёт или электронные таблицы)
8) недостаток такого подхода в том, что он *косвенный*, то есть требует преобразований; поэтому есть неплохой шанс ошибиться при выводе формулы и поэтому получить неверный результат
9) если вы видите, что программа работает очень долго (результата не дождаться), нужно применять метод динамического программирования, сохраняя в массиве все промежуточные результаты (см. подробности в решении задачи P-01)
10) Ответ: 1957585.

Ещё пример задания:

P-01. Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 1$ при $n = 1$

$F(n) = n + 2 + F(n–1)$, если n чётно,

$F(n) = 2 \cdot F(n–2)$, если n нечётно.

Чему равно значение функции $F(24)$? Для выполнения задания можно также написать программу или воспользоваться редактором электронных таблиц.

Решение (ручной счёт от последнего значения):

1) чтобы вычислить $F(24)$, используем формулу для чётных n:

$F(24) = 24 + 2 + F(23)$

2) нам неизвестно значение $F(23)$, поэтому, применяя формулу для нечётных n, находим
$F(23) = 2 \cdot F(21)$
3) далее так же придётся написать формулы для вычисления $F(21)$, $F(19)$, …, $F(3)$, и в конце мы получим
$F(3) = 2 \cdot F(1)$
4) значение $F(1) = 1$ нам задано, подставляя его в предыдущую формулу, находим
$F(3) = 2 \cdot F(1) = 2$
5) теперь значение подставляем в формулу для $F(5)$, потом найденное значение $F(5)$ – в формулу для $F(7)$, и т.д.
6) после продолжительных вычислений получим $F(24) = 2074$
7) Ответ: 2074.

Решение (ручной счёт от первого значения):

1) примерно то же самое можно сделать, начиная вычисления с малых значений n
2) сразу записываем в таблицу известное значение $F(1) = 1$, затем последовательно вычисляем
$F(2) = 2 + 2 + F(1) = 5$ по формуле для чётных n
$F(3) = 2 \cdot F(1) = 2$ по формуле для нечётных n
$F(4) = \ldots$
…
$F(24) = 24 + 2 + F(23) = 2074$
3) результаты вычислений удобно хранить в виде таблицы:

n	1	2	3	4	5	6	7	8	9	…	24
$F(n)$	1	5	2	8	4	12	8	18	16		2074

4) недостаток этого метода в том, что мы вычисляем все значения $F(n)$ подряд, хотя для нахождения $F(24)$ нам не нужны значения $F(n)$ при чётных n
5) Ответ: 2074.

Общий недостаток первых двух методов – **большой объём ручных вычислений**, который приводит к большой вероятности ошибки.

Решение (использование табличного процессора):

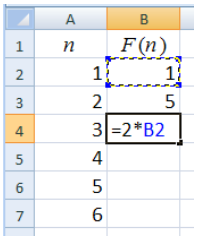
1) для выполнения большого объёма вычислений можно использовать табличный процессор; удобнее строить таблицу из двух столбцов (хотя можно, конечно использовать и две строки) – n и $F(n)$
2) сразу записываем известное значение $F(1) = 1$

	A	B
1	n	$F(n)$
2	1	1
3	2	
4	3	
5	4	
6	5	

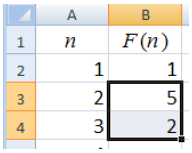
3) в ячейку B3, где вычисляется $F(2)$, записываем формулу для чётных n; здесь соответствующее значение n хранится в A3, а значение $F(n–1) = F(1)$ – в ячейке B2

	A	B
1	n	$F(n)$
2	1	1
3	2	=A3+2+B2
4	3	
5	4	
6	5	
7	6	

4) в ячейку B4 , где вычисляется $F(3)$, записываем формулу для нечётных n; для этой ячейки соответствующее значение $F(n–2) = F(1)$ – в ячейке B2



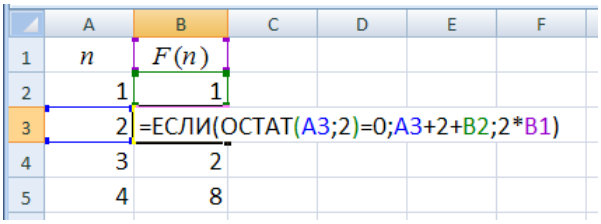
5) поскольку формулы чередуются (для чётных и нечётных n), выделяем **две** введённые формулы и «протягиваем» их, копируя вниз до строки с $n = 24$:



6) напротив значения $n = 24$ считываем ответ – 2074

7) Ответ: 2074.

8) можно обойтись и одной формулой, только в ней придётся использовать функцию ЕСЛИ:



Функция ОСТАТ вычисляет остаток от деления n на 2; если этот остаток равен нулю, то значение n чётное.

Решение (составление программы – рекурсивная функция):

1) составление программы – самый простой и наглядный способ решения задачи; для этого нужно просто реализовать заданные формулы в виде функции на языке программирования;

2) эта задача не должна представлять какой-то сложности; главная проблема – не получить бесконечную рекурсию; для этого рекомендуется сразу в начале функции записать условие окончания рекурсии, которое задаётся условием «$F(n) = 1$ при $n = 1$», и сразу выйти из функции

3) программа на языке Python:

```
def F( n ):
  if n == 1: return 1
  if n % 2 == 0:
    return n + 2 + F(n-1)
  else:
    return 2 * F(n-2)

print( F(24) )
```

4) программа на языке Паскаль:

```
function F( n: integer ): integer;
begin
  if n = 1 then begin
    Result := 1;
    Exit;
  end;
  if n mod 2 = 0 then
```

```
      Result := n + 2 + F(n-1)
    else
      Result := 2 * F(n-2);
  end;
  begin
    writeln( F(24) )
  end.
```

5) программа на языке C++:

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

int F( int n )
{
  if( n == 1 ) return 1;
  if( n % 2 == 0 )
    return n + 2 + F(n-1);
  else
    return 2 * F(n-2);
}
int main()
{
  cout << F(24);
}
```

Решение (составление программы – динамическое программирование):

1) если рекурсивная функция вызывает сама себя несколько раз (для рассматриваемой задачи это не так, но вполне может быть…), при больших значениях аргумента функция $F(n)$ может вычисляться очень (и даже недопустимо!) долго – это один из недостатков рекурсивной реализации
2) во многих случаях достаточно просто заменить рекурсивную функцию нерекурсивной (в принципе, это можно сделать всегда, вопрос лишь в том, какие усилия придётся для этого приложить)
3) один из простых приёмов – использование динамического программирования, которое позволяет свести вычисление значения функции, заданной рекурсивно, к заполнению массива (таблицы) – так же, как при ручном счёте (этот метод решения задачи описан выше)
4) пусть известно значение n; построим массив **a**, так чтобы значение элемента **a[n]** совпадало со значением $F(n)$
5) учитывая, что в Python и в C++ нумерация элементов массива начинается с нуля, будем использовать фиктивный нулевой элемент – не будем его использовать вообще; согласно условию, в первый элемент нужно записать число 1 («$F(n) = 1$ при $n = 1$»):

```
    a = [0, 1]
```

6) затем перебираем в цикле все значения индексов элементов, начиная с 2 и до n включительно, вычисляя для каждого значение функции; фактически сначала вычисляется $F(2)$ и записывается в элемент массива **a[2]**, затем находим $F(3)$ и записываем в элемент массива **a[3]** и т.д. :

```
for i in range(2, n+1):
  if i % 2 == 0:
    a.append( i + 2 + a[i-1] )
  else:
    a.append( 2*a[i-2] )
```

7) заметьте, что вместо рекурсивных вызовов здесь используются уже готовые значения $F(n)$ для меньших значений n, ранее записанные в массив **a** (см. выделение синим цветом)
8) теперь можно построить функцию, которая возвращает значение **a[n]**:

```
def F( n ):
  a = [0, 1]
  for i in range(2, n+1):
    if i % 2 == 0:
      a.append( i + 2 + a[i-1] )
    else:
      a.append( 2*a[i-2] )
  return a[n]
```

9) при вызове

```
print( F(24) )
```

эта функция возвращает тот же результат 2074, что и рекурсивная функция
10) Ответ: 2074.
11) преимущество такого подхода проявится, если функция вызывает сама себя несколько раз; в этом случае каждое значение будет вычисляться только один раз и сохраняться в массиве; когда это значение понадобится снова, оно будет взято сразу из массива в готовом виде (этот приём иногда называют *мемоизацией* – запоминанием)
12) учитывая, что преобразования программы не слишком простые, рекомендуется использовать такой метод на экзамене только тогда, когда «лобовое» решение работает слишком медленно (вы не смогли дождаться результата); помните, что любой шаг в сторону от простейшего решения может стать источником дополнительных ошибок
13) решение на языке Паскаль (элементы массива нумеруются с единицы):

```
function F( n: integer ): integer;
var a: array[1..100] of integer;
    i: integer;
begin
  a[1] := 1;
  for i:=2 to n do
    if i mod 2 = 0 then
      a[i] := i + 2 + a[i-1]
    else
      a[i] := 2 * a[i-2];
  Result := a[n];
end;
begin
  writeln( F(24) )
end.
```

14) решение на языке C++

```
#include <iostream>
using namespace std;
int F( int n )
{
  int i, a[100] = {};
  a[1] = 1;
  for( i = 2; i <= n; i++ )
    if( i % 2 == 0 )
      a[i] = i + 2 + a[i-1];
    else
```

```
      a[i] = 2 * a[i-2];
  return a[n];
}
int main()
{
  cout << F(24);
}
```

Решение (составление программы – вообще без массива и рекурсии):

1) описанный далее метод решения задачи, в принципе, можно использовать, но при сдаче экзамена он **не имеет никаких преимуществ** перед методом динамического программирования, и в то же время повышает ваши шансы сделать ошибку; поэтому подумайте дважды (трижды, четырежды, …), прежде чем попытаться «блеснуть мастерством», к сожалению, при текущей модели экзамена это никто не оценит…
2) в данной задаче можно заметить, что очередное значение $F(n)$ зависит только от $F(n–1)$ или от $F(n–2)$; следовательно, знания значений $F(n–1)$ и $F(n–2)$ всегда будет достаточно для вычисления $F(n)$
3) поэтому можно вычислить $F(n)$, вообще не используя массив, нужно только хранить значения $F(n–1)$ и $F(n–2)$ в переменных
4) в приведённой далее программе переменные **fn1** и **fn2** хранят соответственно значения $F(n–1)$ и $F(n–2)$; начальное значение **fn1** должно быть равно 1 ($F(1) = 1$), а начальное значение **fn2** можно выбрать любым (например, 0), потому что на первой итерации цикла (при вычислении $F(2)$) оно все равно не используется

```
def F( n ):
  fn2 = 0  # эту строку можно удалить
  fn1 = 1
  for i in range(2, n+1):
    if i % 2 == 0:
      fn = i + 2 + fn1
    else:
      fn = 2*fn2
    fn2, fn1 = fn1, fn
  return f
```

Очередное значение $F(n)$ хранится в переменной **fn**. Строка

```
fn2, fn1 = fn1, fn
```

выполняет сдвиг переменных при переходе к следующему значению n: $F(n–1)$ записывается на место $F(n–2)$, а $F(n)$ – на место $F(n–1)$. При вызове

```
print( F(24) )
```

эта функция возвращает тот же результат 2074, что и рекурсивная функция

5) Ответ: 2074.
6) решение на языке Паскаль:

```
function F( n: integer ): integer;
var fn1, fn2, fn, i: integer;
begin
  fn1 := 1;
  for i:=2 to n do begin
    if i mod 2 = 0 then
      fn := i + 2 + fn1
    else
```

```
      fn := 2 * fn2;
    fn2 := fn1;
    fn1 := fn;
  end;
  Result := fn;
end;
begin
  writeln( F(24) )
end.
```

7) решение на языке C++

```
#include <iostream>
using namespace std;
int F( int n )
{
  int i, fn, fn1, fn2;
  fn1 = 1;
  for( i = 2; i <= n; i++ ) {
    if( i % 2 == 0 )
      fn = i + 2 + fn1;
    else
      fn = 2 * fn2;
    fn2 = fn1;
    fn1 = fn;
    }
  return fn;
}
int main()
{
  cout << F(24);
}
```

Задачи для тренировки:

1) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

   $F(n) = 1$ при $n = 1$

   $F(n) = 2 \cdot F(n–1) + n + 3$, если $n > 1$

   Чему равно значение функции $F(19)$?

2) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

   $F(n) = 3$ при $n = 1$

   $F(n) = 2 \cdot F(n–1) – n + 1$, если $n > 1$

   Чему равно значение функции $F(21)$?

3) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

   $F(n) = 2$ при $n = 1$

   $F(n) = F(n–1) + 5n^2$, если $n > 1$

   Чему равно значение функции $F(39)$?

4) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

   $F(n) = 2$ при $n \le 1$

   $F(n) = F(n–1) + F(n–2) + 2n + 4$, если $n > 1$

   Чему равно значение функции $F(25)$?

5) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 3$ при $n \le 1$
$F(n) = F(n–1) + 2 \cdot F(n–2) – 5$, если $n > 1$

Чему равно значение функции $F(22)$?

6) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 2$ при $n \le 1$
$F(n) = F(n–1) + F(n–2) + 4n$, если $n > 1$

Чему равно значение функции $F(24)$?

7) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = n$ при $n > 15$
$F(n) = 2 \cdot F(n+1) + 5n + 2$, если $n \le 15$

Чему равно значение функции $F(2)$?

8) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = n$ при $n > 18$
$F(n) = 3 \cdot F(n+1) + n + 8$, если $n \le 18$

Чему равно значение функции $F(9)$?

9) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = n – 3$ при $n > 16$
$F(n) = 2 \cdot F(n+1) + 2n + 3$, если $n \le 16$

Чему равно значение функции $F(2)$?

10) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 2n – 5$ при $n > 12$
$F(n) = 2 \cdot F(n+2) + n – 4$, если $n \le 12$

Чему равно значение функции $F(1)$?

11) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 1$ при $n = 1$
$F(n) = 2 \cdot F(n–1)$, если n чётно,
$F(n) = 5n + F(n–2)$, если n нечётно.

Чему равно значение функции $F(64)$?

12) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = n$ при $n < 1$
$F(n) = n + 3 \cdot F(n–3)$, если n чётно,
$F(n) = 5n + 2 \cdot F(n–5)$, если n нечётно.

Чему равно значение функции $F(30)$?

13) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 2 \cdot n$ при $n < 3$
$F(n) = 3n + 5 + F(n–2)$, если n чётно,
$F(n) = n + 2 \cdot F(n–6)$, если n нечётно.

Чему равно значение функции $F(61)$?

14) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = –n$ при $n < 0$
$F(n) = 2n + 1 + F(n–3)$, если n чётно,
$F(n) = 4n + 2 \cdot F(n–4)$, если n нечётно.

Чему равно значение функции $F(33)$?

15) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 5–n$ при $n < 5$
$F(n) = 4 \cdot (n – 5) \cdot F(n–5)$, если n делится на 3,
$F(n) = 3n + 2 \cdot F(n–1) + F(n–2)$, если n не делится на 3.

Чему равно значение функции $F(20)$?

16) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = 1+2n$ при $n < 5$

$F(n) = 2\cdot(n + 1)\cdot F(n–2)$, если n делится на 3,

$F(n) = 2\cdot n + 1 + F(n–1) + 2\cdot F(n–2)$, если n не делится на 3.

Чему равно значение функции $F(15)$?

17) Алгоритм вычисления функции $F(n)$ задан следующими соотношениями:

$F(n) = n + 3$ при $n < 3$

$F(n) = (n + 2)\cdot F(n–4)$, если n делится на 3,

$F(n) = n + F(n–1) + 2\cdot F(n–2)$, если n не делится на 3.

Чему равно значение функции $F(20)$?

18) Алгоритм вычисления функций $F(n)$ и $G(n)$ задан следующими соотношениями:

$F(1) = G(1) = 1$

$F(n) = 2\cdot F(n–1) + G(n–1) – 2$, если $n > 1$

$G(n) = F(n–1) +2\cdot G(n–1)$, если $n > 1$

Чему равно значение $F(14) + G(14)$?

19) Алгоритм вычисления функций $F(n)$ и $G(n)$ задан следующими соотношениями:

$F(1) = G(1) = 1$

$F(n) = 2\cdot F(n–1) + G(n–1) – 2n$, если $n > 1$

$G(n) = F(n–1) +2\cdot G(n–1) + n$, если $n > 1$

Чему равно значение $F(14) + G(14)$?

20) Алгоритм вычисления функций $F(n)$ и $G(n)$ задан следующими соотношениями:

$F(1) = G(1) = 1$

$F(n) = 3\cdot F(n–1) + G(n–1) – n + 5$, если $n > 1$

$G(n) = F(n–1) + 3\cdot G(n–1) – 3\cdot n$, если $n > 1$

Чему равно значение $F(14) + G(14)$?

21) Определите, сколько символов * выведет эта процедура при вызове F(28):

Python	Паскаль	C++

`def F( n ):`
`  print('*')`
`  if n >= 1:`
`  print('*')`
`  F(n-1)`
`  F(n-2)`
	`procedure F( n: integer );`
	`begin`
	`  write('*');`
	`  if n >= 1 then begin`
	`  write('*');`
	`  F(n-1);`
	`  F(n-2);`
	`  end;`
	`end;`
		`void F( int n )`
		`{`
		`  cout << '*';`
		`  if( n >= 1 ) {`
		`  cout << '*';`
		`  F(n-1);`
		`  F(n-2);`
		`  }`
		`}`

22) Определите, сколько символов * выведет эта процедура при вызове F(35):

Python	Паскаль	C++

`def F( n ):`
`  print('*')`
`  if n >= 1:`
`  print('*')`
`  F(n-1)`
`  F(n-2)`
`  print('*')`
	`procedure F( n: integer );`
	`begin`
	`  write('*');`
	`  if n >= 1 then begin`
	`  write('*');`
	`  F(n-1);`
	`  F(n-2);`
	`  write('*');`
	`  end;`
	`end;`
		`void F( int n )`
		`{`
		`  cout << '*';`
		`  if( n >= 1 ) {`
		`  cout << '*';`
		`  F(n-1);`
		`  F(n-2);`
		`  cout << '*';`
		`  }`
		`}`

23) Определите, сколько символов * выведет эта процедура при вызове F(40):

Python	Паскаль	C++

def F(n):
  print('*')
  if n >= 1:
  print('*')
  F(n-1)
  F(n-3)
  print('*')
	procedure F(n: integer);
	begin
	  write('*');
	  if n >= 1 then begin
	  write('*');
	  F(n-1);
	  F(n-3);
	  write('*');
	  end;
	end;
		void F(int n)
		{
		  cout << '*';
		  if(n >= 1) {
		  cout << '*';
		  F(n-1);
		  F(n-3);
		  cout << '*';
		  }
		}

24) Определите, сколько символов * выведет эта процедура при вызове F(280):

Python	Паскаль	C++

def F(n):
  print('*')
  if n >= 1:
  print('*')
  F(n-1)
  F(n//3)
  print('*')
	procedure F(n: integer);
	begin
	  write('*');
	  if n >= 1 then begin
	  write('*');
	  F(n-1);
	  F(n div 3);
	  write('*');
	  end;
	end;
		void F(int n)
		{
		  cout << '*';
		  if(n >= 1) {
		  cout << '*';
		  F(n-1);
		  F(n/3);
		  cout << '*';
		  }
		}

25) Определите, сколько символов * выведет эта процедура при вызове F(140):

Python	Паскаль	C++

def F(n):
  print('*')
  if n >= 1:
  print('*')
  F(n-1)
  F(n//2)
	procedure F(n: integer);
	begin
	  write('*');
	  if n >= 1 then begin
	  write('*');
	  F(n-1);
	  F(n div 2);
	  end;
	end;
		void F(int n)
		{
		  cout << '*';
		  if(n >= 1) {
		  cout << '*';
		  F(n-1);
		  F(n/2);
		  }
		}

26) Определите наименьшее значение n, при котором сумма чисел, которые будут выведены при вызове $F(n)$, будет больше 1000000. Запишите в ответе сначала найденное значение n, а затем через пробел – соответствующую сумму выведенных чисел.

Python	Паскаль	C++

def F(n):
  print(n+1)
  if n > 1:
  print(n+5)
  F(n-1)
  F(n-2)
	procedure F
	  (n: integer);
	begin
	  writeln(n+1);
	  if n > 1 then begin
	  writeln(n+5);
	  F(n-1);
	  F(n-2);
	  end;
	end;
		void F(int n)
		{
		  cout << n+1 << endl;
		  if(n > 1) {
		  cout << n+5 << endl;
		  F(n-1);
		  F(n-2);
		  }
		}

27) Определите наименьшее значение n, при котором сумма чисел, которые будут выведены при вызове $F(n)$, будет больше 1000000. Запишите в ответе сначала найденное значение n, а затем через пробел – соответствующую сумму выведенных чисел.

Python	Паскаль	C++

def F(n):
  print(n+1)
  if n > 1:
  print(2*n)
  F(n-1)
  F(n-3)
	procedure F
	  (n: integer);
	begin
	  writeln(n+1);
	  if n > 1 then begin
	  writeln(2*n);
	  F(n-1);
	  F(n-3);
	  end;
	end;
		void F(int n)
		{
		  cout << n+1 << endl;
		  if(n > 1) {
		  cout << 2*n << endl;
		  F(n-1);
		  F(n-3);
		  }
		}

28) Определите наименьшее значение n, при котором сумма чисел, которые будут выведены при вызове $F(n)$, будет больше 5000000. Запишите в ответе сначала найденное значение n, а затем через пробел – соответствующую сумму выведенных чисел.

Python	Паскаль	C++

def F(n):
  print(2*n+1)
  if n > 1:
  print(3*n-8)
  F(n-1)
  F(n-4)
	procedure F
	  (n: integer);
	begin
	  writeln(2*n+1);
	  if n > 1 then begin
	  writeln(3*n-8);
	  F(n-1);
	  F(n-4);
	  end;
	end;
		void F(int n)
		{
		  cout << 2*n+1 << endl;
		  if(n > 1) {
		  cout << 3*n-8
		  << endl;
		  F(n-1);
		  F(n-4);
		  }
		}

29) Определите наименьшее значение n, при котором сумма чисел, которые будут выведены при вызове $F(n)$, будет больше 3200000. Запишите в ответе сначала найденное значение n, а затем через пробел – соответствующую сумму выведенных чисел.

Python	Паскаль	C++

def F(n):
  print(n-5)
  if n > 1:
  print(n+8)
  F(n-2)
  F(n-3)
	procedure F
	  (n: integer);
	begin
	  writeln(n-5);
	  if n > 1 then begin
	  writeln(n+8);
	  F(n-2);
	  F(n-3);
	  end;
	end;
		void F(int n)
		{
		  cout << n-5 << endl;
		  if(n > 1) {
		  cout << n+8 << endl;
		  F(n-2);
		  F(n-3);
		  }
		}

30) Определите наименьшее значение n, при котором сумма чисел, которые будут выведены при вызове $F(n)$, будет больше 3200000. Запишите в ответе сначала найденное значение n, а затем через пробел – соответствующую сумму выведенных чисел.

Python	Паскаль	C++

def F(n):
  print(n*n)
  if n > 1:
	procedure F
	  (n: integer);
	begin
		void F(int n)
		{
		  cout << n*n << endl;

```
    print(2*n+1)
    F(n-2)
    F(n//3)
```

```
  writeln(n*n);
  if n > 1 then begin
    writeln(2*n+1);
    F(n-2);
    F(n div 3);
  end;
end;
```

```
  if( n > 1 ) {
    cout << 2*n+1 << endl;
    F(n-2);
    F(n/3);
    }
}
```