Лабораторная работа 34. Рекурсия и рекурсивные алгоритмы

Цель работы: изучить понятие, виды рекурсии и рекурсивную триаду, научиться разрабатывать рекурсивную триаду при решении задач на языке C++.

При выполнении лабораторной работы для каждого задания требуется написать программу на языке C++, которая получает на входе числовые данные, выполняет их обработку в соответствии с требованиями задания и выводит результат на экран. Для обработки данных необходимо реализовать рекурсивную функцию. Ввод данных осуществляется с клавиатуры с учетом требований к входным данным, содержащихся в постановке задачи (ввод данных сопровождайте диалогом). Ограничениями на входные данные является допустимый диапазон значений используемых числовых типов в языке C++.

Теоретические сведения.

Ознакомьтесь с материалом лекции.

Задания к лабораторной работе.

Составьте рекурсивную функцию для решения задачи (по вариантам).

1. Найдите сумму всех трехзначных чисел, кратных 25.
2. Переведите натуральное число N в восьмеричную систему счисления.
3. Найдите n-ый член геометрической прогрессии, заданной первым членом и знаменателем.
4. Найдите сумму первых n четных натуральных чисел
5. Найдите n-ый член арифметической прогрессии, заданной первым членом и разностью.
6. Определите закономерность формирования членов последовательности $1, 1, \sqrt{2}, \sqrt{1+\sqrt{2}}, \sqrt{\sqrt{2}+\sqrt{1+\sqrt{2}}}, \dots$. Найдите N-ый член последовательности, сократив количество рекурсивных вызовов.
7. Найдите сумму первых натуральных чисел, оканчивающихся цифрой 5.
8. Найдите сумму первых n натуральных чисел, оканчивающихся цифрой 5.
9. Функция определена на полуинтервале $[0;2)$ следующим образом: $f(x)=\sqrt{4-x^2}$. Выполните ее периодическое продолжение на множество действительных чисел. Найдите значение полученной функции для данного x.
10. Переведите натуральное число N в двоичную систему счисления.
11. Разработайте рекурсивную функцию, подсчитывающую количество способов разбиения выпуклого многоугольника на треугольники непересекающимися диагоналями.
12. Определите закономерность формирования членов последовательности $1, 1, 2, 5, 29, \dots$. Найдите N-ый член последовательности, сократив количество рекурсивных вызовов.

Указания к выполнению работы.

Каждое задание необходимо решить в соответствии с изученными рекурсивными методами решения задач и методами обработки числовых данных в языке C++. Перед реализацией кода каждой задачи необходимо разработать рекурсивную триаду в соответствии с постановкой задачи: выполнить параметризацию, выделить базу и оформить декомпозицию рекурсии. Этапы рекурсивной триады необходимо отразить в математической модели к отчету, выполнив обоснование декомпозиции. Программу для решения каждого задания необходимо разработать методом процедурной абстракции, используя рекурсивные функции. Этапы сопроводить комментариями в коде.

Следует реализовать каждое задание в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;
- разработать контрольный тест к программе;
- отладить программу;
- представить отчет по работе.

Требования к отчету.

Отчет по лабораторной работе должен соответствовать следующей структуре.

- Титульный лист.
- Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.
- Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
- Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).
- Листинг программы. Подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio.
- Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
- Выводы по лабораторной работе.
- Ответы на контрольные вопросы.

Рекурсивный алгоритм – это алгоритм, в описании которого прямо или косвенно содержится обращение к самому себе. В технике процедурного программирования данное понятие распространяется на функцию, которая реализует решение отдельного блока задачи посредством вызова из своего тела других функций, в том числе и себя самой. Если при этом на очередном этапе работы функция организует обращение к самой себе, то такая функция является рекурсивной.

Прямое обращение функции к самой себе предполагает, что в теле функции содержится вызов этой же функции, но с другим набором фактических параметров. Такой способ организации работы называется прямой рекурсией. Например, чтобы найти сумму первых n натуральных чисел, надо сумму первых (n-1) чисел сложить с числом n, то есть имеет место зависимость: $S_n=S_{n-1}+n$. Вычисление происходит с помощью аналогичных рассуждений. Такая цепочка взаимных обращений в конечном итоге сведется к вычислению суммы одного первого элемента, которая равна самому элементу.

При косвенном обращении функция содержит вызовы других функций из своего тела. При этом одна или несколько из вызываемых функций на определенном этапе обращаются к исходной функции с измененным набором входных параметров. Такая организация обращений называется косвенной рекурсией. Например, поиск максимального элемента в массиве размера n можно осуществлять как поиск максимума из двух чисел: одно их них – это последний элемент массива, а другое является максимальным элементом в массиве размера (n-1). Для нахождения максимального элемента массива размера (n-1) применяются аналогичные рассуждения. В итоге решение сводится к поиску максимального из первых двух элементов массива.

Рекурсивный метод в программировании предполагает разработку решения задачи, основываясь на свойствах рекурсивности отдельных объектов или закономерностей. При этом исходная задача сводится к решению аналогичных подзадач, которые являются более простыми и отличаются другим набором параметров.

Разработке рекурсивных алгоритмов предшествует **рекурсивная триада** – этапы моделирования задачи, на которых определяется набор параметров и соотношений между ними. Рекурсивную триаду составляют параметризация, выделение базы и декомпозиция.

На этапе параметризации из постановки задачи выделяются параметры, которые описывают исходные данные. При этом некоторые дальнейшие разработки решения могут требовать введения дополнительных параметров, которые не оговорены в условии, но используются при составлении зависимостей. Необходимость в дополнительных параметрах часто возникает также при решении задач оптимизации рекурсивных алгоритмов, в ходе которых сокращается их временная сложность.

Выделение базы рекурсии предполагает нахождение в решаемой задаче тривиальных случаев, результат для которых очевиден и не требует проведения расчетов. Верно найденная база рекурсии обеспечивает завершенность рекурсивных обращений, которые в конечном итоге сводятся к базовому случаю. Переопределение базы или ее динамическое расширение в ходе решения задачи часто позволяют оптимизировать рекурсивный алгоритм за счет достижения базового случая за более короткий путь обращений.

Декомпозиция представляет собой сведение общего случая к более простым подзадачам, которые отличаются от исходной задачи набором входных данных. Декомпозиционные зависимости описывают не только связь между задачей и подзадачами, но и характер изменения значений параметров на очередном шаге. От выбранных отношений зависит трудоемкость алгоритма, так как для одной и той же задачи могут быть составлены различные зависимости. Пересмотр отношений декомпозиции целесообразно проводить

комплексно, то есть параллельно с корректировкой параметров и анализом базовых случаев.

Анализ трудоемкости рекурсивных алгоритмов методом подсчета вершин дерева рекурсии

Рекурсивные алгоритмы относятся к классу алгоритмов с высокой ресурсоемкостью, так как при большом количестве самовызовов рекурсивных функций происходит быстрое заполнение стековой области. Кроме того, организация хранения и закрытия очередного слоя рекурсивного стека являются дополнительными операциями, требующими временных затрат. На трудоемкость рекурсивных алгоритмов влияет и количество передаваемых функцией параметров.

Рассмотрим один из методов анализа трудоемкости рекурсивного алгоритма, который строится на основе подсчета вершин рекурсивного дерева. Для оценки трудоемкости рекурсивных алгоритмов строится **полное дерево рекурсии**. Оно представляет собой граф, вершинами которого являются наборы фактических параметров при всех вызовах функции, начиная с первого обращения к ней, а ребрами – пары таких наборов, соответствующих взаимным вызовам. При этом вершины дерева рекурсии соответствуют фактическим вызовам рекурсивных функций. Следует заметить, что одни и те же наборы параметров могут соответствовать разным вершинам дерева. Корень полного дерева рекурсивных вызовов – это вершина полного дерева рекурсии, соответствующая начальному обращению к функции.

Важной характеристикой рекурсивного алгоритма является **глубина рекурсивных вызовов** – наибольшее одновременное количество рекурсивных обращений функции, определяющее максимальное количество слоев рекурсивного стека, в котором осуществляется хранение отложенных вычислений. Количество элементов полных рекурсивных обращений всегда не меньше глубины рекурсивных вызовов. При разработке рекурсивных программ необходимо учитывать, что глубина рекурсивных вызовов не должна превосходить максимального размера стека используемой вычислительной среды.

При этом **объем рекурсии -** это одна из характеристик сложности рекурсивных вычислений для конкретного набора параметров, представляющая собой количество вершин полного рекурсивного дерева без единицы.

Будем использовать следующие обозначения для конкретного входного параметра D:

R(D) – общее число вершин дерева рекурсии,

$R_V(D)$ – объем рекурсии без листьев (внутренние вершины),

$R_L(D)$ – количество листьев дерева рекурсии,

$H_R(D)$ – глубина рекурсии.

Например, для вычисления n -го члена последовательности Фибоначчи разработана следующая рекурсивная функция:

```
int Fib(int n){ //n – номер члена последовательности
  if(n<3) return 1; //база рекурсии
  return Fib(n-1)+Fib(n-2); //декомпозиция
}
```

Тогда полное дерево рекурсии для вычисления пятого члена последовательности Фибоначчи будет иметь вид (рис. 34.1):

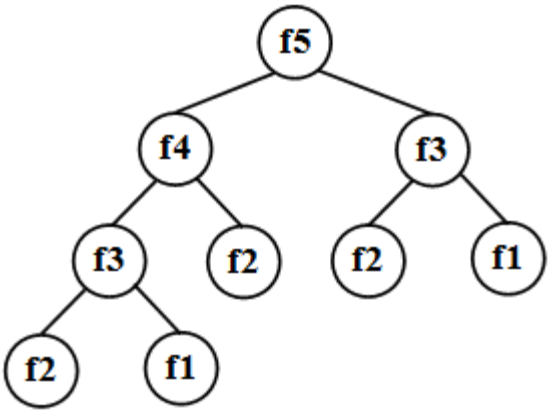

Рис. 34.1. Полное дерево рекурсии для пятого члена последовательности Фибоначчи

Характеристиками рассматриваемого метода оценки алгоритма будут следующие величины.

D = 5	D = n
R(D)=9	R(D)=2fn-1
$R_V(D)=4$	$R_V(D)=fn-1$
$R_L(D)=5$	$R_L(D)=fn$
$H_R(D)=4$	$H_R(D)=n-1$

Пример 1. Задача о разрезании прямоугольника на квадраты.

Дан прямоугольник, стороны которого выражены натуральными числами. Разрежьте его на минимальное число квадратов с натуральными сторонами. Найдите число получившихся квадратов.

Разработаем рекурсивную триаду.

Параметризация: m, n – натуральные числа, соответствующие размерам прямоугольника.

База рекурсии: для m=n число получившихся квадратов равно 1, так как данный прямоугольник уже является квадратом.

Декомпозиция: если $m \neq n$, то возможны два случая m < n или m > n. Отрежем от прямоугольника наибольший по площади квадрат с натуральными сторонами. Длина стороны такого квадрата равна наименьшей из сторон прямоугольника. После того, как квадрат будет отрезан, размеры прямоугольника станут следующие: большая сторона уменьшится на длину стороны квадрата, а меньшая не изменится. Число искомых квадратов будет вычисляться как число квадратов, на которые будет разрезан полученный прямоугольник, плюс один (отрезанный квадрат). К получившемуся прямоугольнику применим аналогичные рассуждения: проверим на соответствие базе или перейдем к декомпозиции (рис. 34.2).

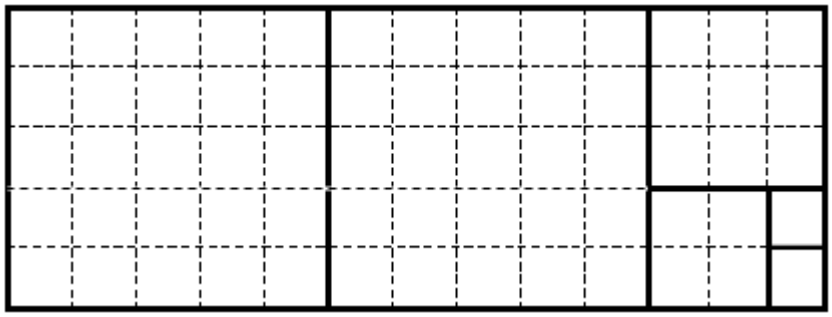

Рис. 34.2. Пример разрезания прямоугольника 13x5 на квадраты

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int kv(int m,int n);

int _tmain(int argc, _TCHAR* argv[]) {
  int a,b,k;
  printf("Введите стороны прямоугольника->");
  scanf("%d%d",&a,&b);
  k = kv(a,b);
  printf("Прямоугольник со сторонами %d и %d можно разрезать
         на %d квадратов",a,b,k);
  system("pause");
  return 0;
}

int kv(int m,int n){ //m,n – стороны прямоугольника
  if(m==n) return 1; //база рекурсии
  if(m>n) return 1+kv(m-n,n); //декомпозиция для m>n
  return 1+kv(m,n-m); //декомпозиция для m<n
}
```

Характеристиками рассматриваемого метода оценки алгоритма будут следующие величины (рис. 34.3).

D = (13, 5)	**D = (m, n), m ≥ n, худший случай**
R(D)=6	R(D)=m
$R_V(D)=4$	$R_V(D)=m-2$
$R_L(D)=1$	$R_L(D)=1$
$H_R(D)=6$	$H_R(D)=m$

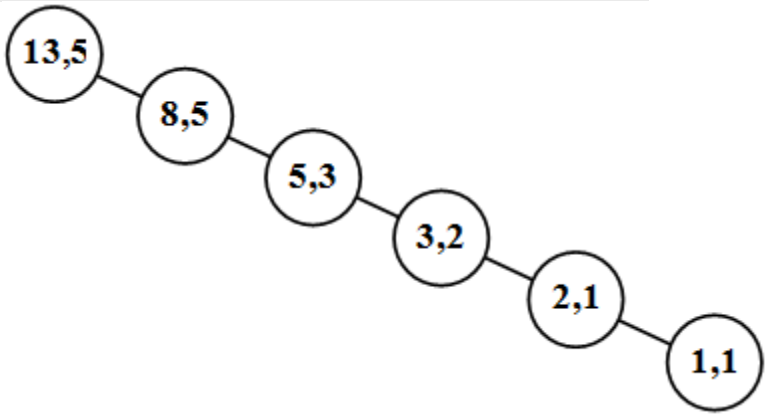

Рис. 34.3. Пример полного дерева рекурсии для разрезания прямоугольника 13x5 на квадраты

Пример 2. Задача о нахождении центра тяжести выпуклого многоугольника.

Выпуклый многоугольник задан на плоскости координатами своих вершин. Найдите его центр тяжести.

Разработаем рекурсивную триаду.

Параметризация: x, y – вещественные массивы, в которых хранятся координаты вершин многоугольника; n – это число вершин многоугольника, по условию задачи, n>1 так как минимальное число вершин имеет двуугольник (отрезок).

База рекурсии: для n=2 в качестве многоугольника рассматривается отрезок, центром тяжести которого является его середина (рис. 4A). При этом середина делит отрезок в отношении 1 : 1. Если координаты концов отрезка заданы как (x_0,y_0) и (x_1,y_1), то координаты середины вычисляются по формуле:

$$cx = \frac{x_0 + x_1}{2}, \quad cy = \frac{y_0 + y_1}{2}.$$

Декомпозиция: если n>2, то рассмотрим последовательное нахождение центров тяжести треугольника, четырехугольника и т.д.

Для n=3 центром тяжести треугольника является точка пересечения его медиан, которая делит каждую медиану в отношении 2 : 1, считая от вершины. Но основание медианы – это середина отрезка, являющегося стороной треугольника. Таким образом, для нахождения центра тяжести треугольника необходимо: найти центр тяжести стороны треугольника (отрезка), затем разделить в отношении 2 : 1, считая от вершины, отрезок, образованный основанием медианы и третьей вершиной (рис. 4B).

Для n=3 центром тяжести четырехугольника является точка, делящая в отношении 3 : 1, считая от вершины, отрезок: он образован центром тяжести треугольника, построенного на трех вершинах, и четвертой вершиной (рис. 4C).

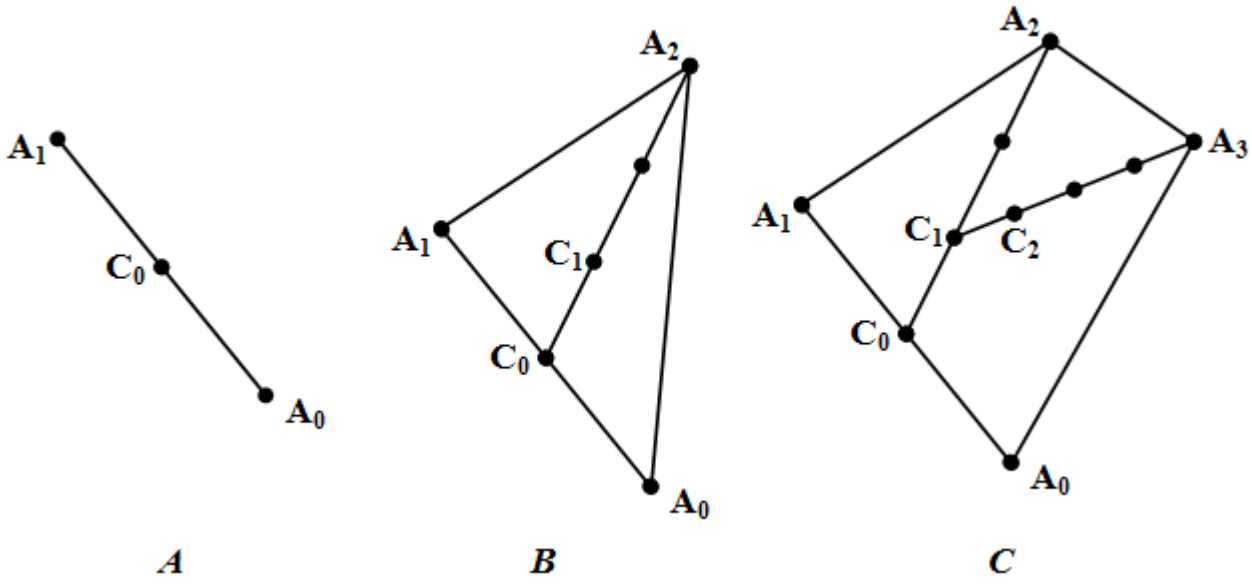

Рис. 34.4. Примеры построения центров тяжести многоугольников

Таким образом, для нахождения центра тяжести n -угольника необходимо разделить в отношении (n-1): 1, считая от вершины, отрезок: он образован центром тяжести (n-1)

-угольника и n -ой вершиной рассматриваемого многоугольника. Если концы отрезка заданы координатами вершины (x_n,y_n) и центра тяжести (n-1) -угольника (cx_{n-1},cy_{n-1}), то при делении отрезка в данном отношении получаем координаты:

$$cx_n = \frac{x_n + (n-1)cx_{n-1}}{n}, \quad cy_n = \frac{y_n + (n-1)cy_{n-1}}{n}$$

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#define max 20
void centr(int n,float *x, float *y, float *c);

int _tmain(int argc, _TCHAR* argv[]){
  int m, i=0;
  FILE *f;
  if ( ( f = fopen("in.txt", "r") ) == NULL )
    perror("in.txt");
  else {
    fscanf(f, "%d",&m);
    printf("\n%d",m);
    if ( m < 2 || m > max ) //вырожденный многоугольник
      printf ("Вырожденный многоугольник");
    else {
     float *px,*py,*pc;
      px = new float[m];
      py = new float[m];
      pc = new float[2];
      pc[0] = pc[1] = 0;
      while(i<m) {
        fscanf(f, "%f %f",&px[i], &py[i]);
        printf("\n%f %f",px[i], py[i]);
        i++;
      }
      centr(m,px,py,pc);
      printf ("\nЦентр тяжести имеет координаты:
              (%.4f, %.4f)",pc[0],pc[1]);
      delete [] pc;
      delete [] py;
      delete [] px;
    }
    fclose(f);
  }
  system("pause");
  return 0;
}

void centr(int n,float *x, float *y, float *c){
//n - количество вершин,
//x,y - координаты вершин,
//c - координаты центра тяжести
  if(n==2){ //база рекурсии
           c[0]=(x[0]+x[1])/2;
           c[1]=(y[0]+y[1])/2;
          }
  if(n>2) { //декомпозиция
           centr(n-1,x,y,c);
           c[0]= (x[n-1] + (n-1)*c[0])/n;
           c[1]= (y[n-1] + (n-1)*c[1])/n;
         }
}
```

Характеристиками рассматриваемого метода оценки алгоритма будут следующие величины.

D = 4 D = n

R(D)=3	R(D)=n-1
R_V(D)=1	R_V(D)=n-3
R_L(D)=1	R_L(D)=1
H_R(D)=3	H_R(D)=n-1

Однако в данном случае для более достоверной оценки необходимо учитывать емкостные характеристики алгоритма.

Пример 3. Задача о разбиении целого на части.

Найдите количество разбиений натурального числа на сумму натуральных слагаемых.

Разбиение подразумевает представление натурального числа в виде суммы натуральных слагаемых, при этом суммы должны отличаться набором чисел, а не их последовательностью. В разбиение также может входить одно число.

Например, разбиение числа 6 будет представлено 11 комбинациями:

```
6
5+1
4+2,  4+1+1
3+3,  3+2+1,  3+1+1+1
2+2+2,  2+2+1+1,  2+1+1+1+1
1+1+1+1+1+1
```

Рассмотрим решение в общем виде. Пусть зависимость R(n,k) вычисляет количество разбиений числа n на сумму слагаемых, не превосходящих k. Опишем свойства R(n,k).

Если в сумме все слагаемые не превосходят 1, то такое представление единственно, то есть R(n,k)=1.

Если рассматриваемое число равно 1, то при любом натуральном значении второго параметра разбиение также единственно: R(n,k)=1.

Если второй параметр превосходит значение первого , то имеет место равенство R(n,k)=R(n,n), так как для представления натурального числа в сумму не могут входить числа, превосходящие его.

Если в сумму входит слагаемое, равное первому параметру, то такое представление также единственно (содержит только это слагаемое), поэтому имеет место равенство: R(n,n)=R(n,n-1)+1.

Осталось рассмотреть случай (n>k). Разобьем все представления числа n на непересекающиеся разложения: в одни обязательно будет входить слагаемое k, а другие суммы не содержат k. Первая группа сумм, содержащая k, эквивалентна зависимости R(n-k,k), что следует после вычитания числа k из каждой суммы. Вторая группа сумм содержит разбиение числа n на слагаемые, каждое из которых не превосходит k-1, то есть число таких представлений равно R(n,k-1). Так как обе группы сумм не пересекаются, то R(n,k)=R(n-k,k)+R(n,k-1).

Разработаем рекурсивную триаду.

Параметризация: Рассмотрим разбиение натурального числа n на сумму таких слагаемых, которые не превосходят натурального числа k.

База рекурсии: исходя из свойств рассмотренной зависимости, выделяются два базовых случая:

при n=1 R(n,k)=1,

при k=1 R(n,k)=1.

Декомпозиция: общий случай задачи сводится к трем случаям, которые и составляют декомпозиционные отношения.

при n=k R(n,k)=R(n,n-1)+1,

при n<k R(n,k)=R(n,n),

при n>k R(n,k)=R(n-k,k)+R(n,k-1).

```
#include "stdafx.h"
#include <iostream>
using namespace std;
unsigned long int Razbienie(unsigned long int n,
                            unsigned long int k);

int _tmain(int argc, _TCHAR* argv[]){
  unsigned long int number, max,num;
  printf ("\nВведите натуральное число: ");
  scanf ("%d", &number);
  printf ("Введите максимальное натуральное слагаемое в
           сумме: ");
  scanf ("%d", &max);
  num=Razbienie(number,max);
  printf ("Число %d можно представить в виде суммы с
           максимальным слагаемым %d.", number, max);
  printf ("\nКоличество разбиений равно %d",num);
  system("pause");
  return 0;
}

unsigned long int Razbienie(unsigned long int n,
                            unsigned long int k){
  if(n==1 || k==1)  return 1;
  if(n<=k)  return Razbienie(n,n-1)+1;
  return Razbienie(n,k-1)+Razbienie(n-k,k);
}
```

Пример 4. Задача о переводе натурального числа в шестнадцатеричную систему счисления.

Дано натуральное число, не выходящее за пределы типа unsigned long. Число представлено в десятичной системе счисления. Переведите его в систему счисления с основанием 16.

Пусть требуется перевести целое число n из десятичной в p -ичную систему счисления (по условию задачи, p = 16), то есть найти такое k, чтобы выполнялось равенство $n_{10}=k_p$.

Параметризация: n – данное натуральное число, p – основание системы счисления.

База рекурсии: на основании правил перевода чисел из десятичной системы в систему счисления с основанием p, деление нацело на основание системы выполняется до тех пор, пока неполное частное не станет равным нулю, то есть: если целая часть частного n и p

равна нулю, то k = n. Данное условие можно реализовать иначе, сравнив n и p: целая часть частного равна нулю, если n < p.

Декомпозиция: в общем случае k формируется из цифр целой части частного n и p, представленной в системе счисления с основанием p, и остатка от деления n на p.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#define maxline 50
void perevod( unsigned long n, unsigned int p,FILE *pf);

int _tmain(int argc, _TCHAR* argv[]){
  unsigned long number10;
  unsigned int osn=16;
  char number16[maxline];
  FILE *f;
  if ((f=fopen("out.txt", "w"))==NULL)
    perror("out.txt");
  else {
    printf ("\nВведите число в десятичной системе: ");
    scanf("%ld", &number10);
    perevod(number10, osn, f);
   fclose(f);
  }
  if ((f=fopen("out.txt", "r"))==NULL)
    perror("out.txt");
  else {
    fscanf(f,"%s",number16);
    printf("\n %ld(10)=%s(16)", number10, number16);
    fclose(f);
  }
  system("pause");
  return 0;
}

void perevod(unsigned long n, unsigned int p, FILE *pf){
  char c;
  unsigned int r;
  if(n >= p) perevod (n/p, p, pf);//декомпозиция
  r=n%p;
  c=r < 10 ? char (r+48) : char (r+55);
  putc(c, pf);
}
```

Ключевые термины

База рекурсии – это тривиальный случай, при котором решение задачи очевидно, то есть не требуется обращение функции к себе.

Глубина рекурсивных вызовов – это наибольшее одновременное количество рекурсивных обращений функции, определяющее максимальное количество слоев рекурсивного стека.

Декомпозиция – это выражение общего случая через более простые подзадачи с измененными параметрами.

Корень полного дерева рекурсивных вызовов – это вершина полного дерева рекурсии, соответствующая начальному обращению к функции.

Косвенная (взаимная) рекурсия – это последовательность взаимных вызовов нескольких функций, организованная в виде циклического замыкания на тело первоначальной функции, но с иным набором параметров.

Объем рекурсии - это характеристика сложности рекурсивных вычислений для конкретного набора параметров, представляющая собой количество вершин полного рекурсивного дерева без единицы.

Параметризация – это выделение из постановки задачи параметров, которые используются для описания условия задачи и решения.

Полное дерево рекурсии – это граф, вершинами которого являются наборы фактических параметров при всех вызовах функции, начиная с первого обращения к ней, а ребрами – пары таких наборов, соответствующих взаимным вызовам.

Прямая рекурсия – это непосредственное обращение рекурсивной функции к себе, но с иным набором входных данных.

Рекурсивная триада – это этапы решения задач рекурсивным методом.

Рекурсивная функция – это функция, которая в своем теле содержит обращение к самой себе с измененным набором параметров.

Рекурсивный алгоритм – это алгоритм, в определении которого содержится прямой или косвенный вызов этого же алгоритма.

Рекурсия – это определение объекта посредством ссылки на себя.

Краткие итоги

1. Рекурсия характеризуется определением объекта посредством ссылки на себя.
2. Рекурсивные алгоритмы содержат в своем теле прямое или опосредованное обращение с самим себе.
3. Рекурсивные функции содержат в своем теле обращение к самим себе с измененным набором параметров в виде прямой рекурсии. При этом обращение к себе может быть организовано посредством косвенной рекурсии – через цепочку взаимных обращений функций, замыкающихся в итоге на первоначальную функцию.
4. Решение задач рекурсивными способами проводится посредством разработки рекурсивной триады.
5. Целесообразность применения рекурсии в программировании обусловлена спецификой задач, в постановке которых явно или опосредовано указывается на возможность сведения задачи к подзадачам, аналогичным самой задаче.
6. Рекурсивные методы решения задач широко используются при моделировании задач из различных предметных областей.
7. Рекурсивные алгоритмы относятся к ресурсоемким алгоритмам. Для оценки сложности рекурсивных алгоритмов учитывается число вершин полного рекурсивного дерева, количество передаваемых параметров, временные затраты на организацию стековых слоев.