Написать процедуру с одним циклом, без использования меток и рекурсивных процедур для вычисления произведения матриц C = A * B;

// размер матрицы
#define N 1000
void mult(double A[][], double B[][], double C[][])

Написать рекурсивную процедуру без циклов и меток для вычисления произведения матриц C = A * B. Оценить размер памяти, требуемой для вычисления.

// размер матрицы
#define N 1000
void mult(double A[][], double B[][], double C[][]);

Подсказка: в программе может быть несколько вспомогательных рекурсивных процедур.

Написать функцию, находящую разбиение множества мощности 30 на два подмножества таких, что сумма элементов одного подмножества равна сумме элементов второго. Исходное множество S задано массивом, результат – логический массив split той же длины, где ложное значение означает принадлежность первому подможеству, а истинное – второму. Функция возвращает истину, если разбиение существует.

#define N 30
#define bool unsigned int
bool split_eq(int S[], bool split[]);

Заданы две строки A и B, составленные из латинских букв. Написать функцию edit, определяющую кратчайшую последовательность редакторских преобразований, с помощью которой строка A может быть преобразована в строку B. Допустимыми редакторскими преобразованиями являются удаление и вставка символа. Результатом должна быть строка из латинских букв и символов '.' и '/', где буква означает – вставить букву, '.' – оставить как было и сдвинуться вправо, '/' – удалить букву. Например, для строк A=”recursion” и B = “response” результатом может быть “..///.p/..se”.

char * edit(char * A, char * B);

Реализовать функцию Фибоначчи с помощью рекурсии.

$Fib(0) = 1$

$Fib(1) = 1$

$Fib(n) = Fib(n-1) + Fib(n-2), n > 1$

Систематически преобразовать в итеративный вариант.

long Fib(long n);

Реализовать функцию Аккермана с помощью рекурсии:

A(0,m) = m+1

A(n,0) = A(n-1,1)

A(n,m) = A(n-1,A(n,m-1)).

где n,m>0.

Систематически преобразовать в итеративный вариант.

```
long Ack(long n, long m);
```

Написать функию diff, вычисляющую для двоичного дерева, заданного указателем на корень, максимум разностей весов поддеревьев для всех вершин. Вес вершины задается возможно отрицательным атрибутом value. Вес поддерева равен сумме весов всех его вершин.

$$diff\,(T) = \max_{v_1, v_2 \in T} (|\, weight(v_1) - weight(v_2)\,|)$$

$$weight(T) = \sum_{v \in T} v.value$$

```
struct STree
{
    int value;
    struct STree * left, * right;
}
int diff(struct STree * t);
```

Написать функцию calc для вычисления следующей формулы

$$\sum_{n=1}^{M} \frac{3^n}{\sum_{k=0}^{n} C_n^k}$$

```
double calc(int M);
```

Написать функцию delete_3mins, которая удаляет в дереве двоичного поиска три минимальных элемента. В случае, если в дереве не более трёх вершин, процедура должна возвращать NULL.

```
struct STree
{
    int value;
    struct STree * left, * right;
}
struct STree * delete_3mins(struct STree * t);
```

Единичным n-мерным кубом называется граф, вершинами которого являются последовательности из 0 и 1 длины n, и две вершины соединены ребром тогда и только тогда, когда они отличаются в единственном разряде. Написать фукцию cube_cycle, которая возвращает гамильтонов цикл в единичном n-мерном кубе, т.е. цикл, проходящий через все вершины ровно по одному разу. Цикл задаётся массивом длины 2^n, i-й элемент которого является номером изменяемого на i-ом шаге разряда.

```
int * cube_cycle(int n);
```

Задан набор из n костей домино, в котором некоторые варианты костей могут быть представлены несколько раз, а некоторые отсутствовать. Написать функцию is_coherent , которая проверяет возможно ли «выложить» все кости набора?

```
struct Domino
{
    int left, right;
}
#define bool unsigned int
bool  is_coherent(struct Domino dices[], int n);
```

Выражение задано в виде рекурсивной ссылочной структуры. Написать функцию expr_to_string, переводящую это представление в текстовую запись выражения в инфиксной записи с минимальным количеством скобок и с учетом приоритета операций.

```
char * expr_to_string(stuct expr * e);
```

```
struct expr {
  unsigned char code; // x, 0, b , u
  union {
    float value;
    unsigned char name[8];
    struct {
      unsigned char op; // -
      struct expr * arg;
    } unop;
    struct {
      unsigned char op; // +, -, *, /
      struct expr * left, * right;
    } binop;
  } case;
};
```

Выражение задано в виде рекурсивной ссылочной структуры. Написать функцию expr_simplify, упрощающую выражение согласно следующих аксиом:

- e + 0 = 0 + e = e – 0 = e
- 0 – e = -e
- $e_1 + (-e_2) = (-e_2) + e_1 = e_1 – e_2$
- e – e = 0
- - - e = e
- 0 * e = e * 0 = 0
- 1 * e = e * 1 = e / 1= e
- e / e = 1
- 1 / (1 / e) = e

struct expr * expr_simplify(stuct expr * e);

```
struct expr {
  unsigned char code; // x, 0, b , u
  union {
    float value;
    unsigned char name[8];
    struct {
      unsigned char op; // -
      struct expr * arg;
    } unop;
    struct {
      unsigned char op; // +, -, *, /
      struct expr * left, * right;
    } binop;
  } case;
};
```

Выражение задано в виде рекурсивной ссылочной структуры. Написать функцию expr_eval, вычисляющую выражение с контролем переполнения/деления на ноль. Значения переменных заданы всюду определённой функцией mem, например,

mem(“x1”) → 12.3

float expr_eval(stuct expr * e, float * mem());

```
struct expr {
  unsigned char code; // x, 0, b , u
  union {
    float value;
    unsigned char name[8];
    struct {
      unsigned char op; // -
      struct expr * arg;
    } unop;
    struct {
      unsigned char op; // +, -, *, /
      struct expr * left, * right;
    } binop;
  } case;
};
```

Написать процедуру, вычисляющую логическое выражение (0 - ложь, 1 - истина, ! – отрицание, & - конъюнкция, | - дизъюнкция, ^ - исключающее ИЛИ), заданное в обратной польской записи, с контролем правильности записи. Например, выражение “(0 | 1) & (1 & !0)” в обратной польской записи имеет вид “0 1 | 1 0 ! &” а выражения в польской записи “0 & 1 |”, “0 1 & 1 1 ! 1 |” и “0 1 + 2 -” – неправильны. В записи выражения допустимы пробелы.

int eval_bool_expr(char * expr);

Написать процедуру, печатающую все последовательности длины n из символов {a,b,c}.

void print_all_abc(int n);

Написать процедуру, печатающую все последовательности из 0 и 1 длины n, содержащие одинаковое количество тех и других:

void print_eq01(int n);

Детерминированный конечный автомат над алфавитом ‘a’..’z’ задан таблицей переходов FSM. Например, для состояния S и символа x следующее состояние определяется как FSM[S][x-‘a’]. Написать функцию fsm_empty, проверяющую пустоту языка автомата:

#define bool unsigned int
bool fsm_empty(int FSM[][]);

Детерминированный конечный автомат над алфавитом ‘a’..’z’ задан таблицей переходов FSM. Например, для состояния S и символа x следующее состояние определяется как FSM[S][x-‘a’]. Написать функцию fsm_acceptn, проверяющую допускает ли этот автомат хотя бы одно слово длины n:

#define bool unsigned int
bool fsm_acceptn(int FSM[][], int n);

Заданы шестицифровая комбинации. Написать процедуру lucky, определяющую можно ли расставить между цифрами символы скобок, операций сложения, вычитания и умножения, чтобы получившиеся арифметические выражения было равно 0. Например, для входа 2 3 1 3 2 2 результатом будет истина, поскольку 23 – 13*2 + 2 = 0

#define bool unsigned int
bool lucky(char a[]);

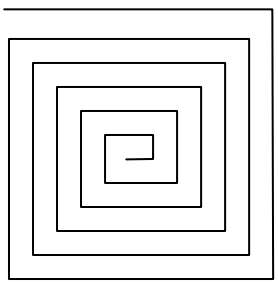
Написать процедуру palindrome_coverage, выдающую для заданной строки её покрытие минимальным количеством непересекающихся палиндромов. Например, для строки 000110010 таким покрытием будет

00
0110
010

Процедура должна возвращать массив из строк, завершающим элементом которого является NULL:

char * * palindrome_coverage(char * s);

Написать процедуру traverse заполнения матрицы B[N*N] элементами матрицы A[N][N] при следующем обходе.



#define N 1000

void traverse(double A[][], double B[][]);

k-диагональная матрица M размера N*N представляется массивом A[k][N], i-я строка которого хранит i-ю диагональ M. Написать процедру mult перемножения двух матриц в таком представлении:

```
#define N 1000
void mult(double A[][], double B[][], double C[][]);
```

Симметричная матрица M размера N*N задаётся одномерным массивом A[N*(N+1}/2], где A = (M_{11}. $M_{21,}$ $M_{22,}$ $M_{31,}$ $M_{32,}$ $M_{33,}$ $M_{41,}$ … $M_{N,N}$). Написать процедуру перемножения двух матриц в таком представлении:

```
#define N 1000
void mult(double A[], double B[], double C[]);
```

Матрица A размера N*N задана одномерным массивом S размера N*N, в котором элементы A расположены по строкам. Написать процедуру поворота матрицы A на 90 градусов по часовой стрелке без использования дополнительных массивов.

```
#define N 1000
void rotate(double S[][]);
```

Задача на засыпку: то же, но не для квадратной матрицы.

Написать процедуру rshift, выполняющую циклический сдвиг элементов массива A длины N на k элементов вправо, не используя дополнительный массив. Например, для входа 1 3 5 7 9 2 4 6 8 10, результатом должно быть 4 6 8 10 1 3 5 7 9 2.

```
#define N 1000
void rshift(double A[], int k);
```

Для заданного символьного массива определить наиболее часто встречающуюся подпоследовательность длины k подряд идущих элементов.

```
char * most_frequent(char s[], int k);
```

Результатом процедуры является ссылка на первое вхождение такой подпоследовательности и NULL в случае её отсутствия.

Написать функцию longest_subseq, вычисляющую для данной последовательности A положительных целых чисел, заданной массивом длины n, наиболее длинную неубывающую подпоследовательность, возможно не подряд идущих элементов. Результирующая последовательность завершается 0.
Например, для последовательности 1 4 8 2 3 6 3 9 2 4 такой подпоследовательностью будет 1 2 3 3 4 0.

```
#define n 1000
int * longest_subseq(int A[]);
```

Массив длины n представляет перестановку чисел от 1 до n Написать функцию decompose, вычисляющую разложение этой перестановки в простые перестановки. Например, для перестановки 3 2 5 4 1 результатом должно быть
3 5 1
2 4

Процедура должна возвращать массив из массивов, завершающим элементом которого является NULL, а каждый из подмассивов заканчивается 0.

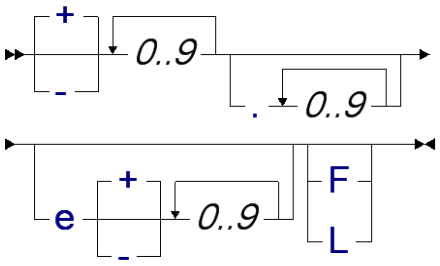
```
#define n 1000
int * * decompose(int A[]);
```

Написать функцию next_perm поиска следующей по алфавиту перестановки, например для 1 2 5 6 4 3 результатом должно быть 1 2 6 3 4 5.

```
#define N 1000
#define bool unsigned char
bool next_perm(double A[]);
```

Процедура возвращает истину, если следующая перестановка существует.

Определить конечный автомат, соответсвующий указанной синтаксической диаграмме вещественного числа в C, и написать функцию parse_float, которой передается строка и которая возвращает истину тогда и только тогда, когда строка-аргумент является правильным представлением вещественного числа в C, а в аргумент f возвращает значение числа.




```
#define bool unsigned char
bool parse_float(char * s, float * f);
```

Написать процедуру simple_sum, печатающую все возможные разложения числа n в сумму различных простых слагаемых.

```
void simple_sum(int n);
```

Написать процедуру horse_step, печатающую обход конем шахматной доски размера nxn, начиная с клетки A1. В каждой клетке конь должен побывать ровно один раз. В случае, если такой обход невозможен, выдать соответствующее сообщение.

```
void horse_step(int n);
```

Написать функцию replace, реализующую *одновременную* замену максимально возможного количества вхождений строки target в строку source на строку subst. Например, для source = “a*aba*b*aba*”, target = “aba”, и subst = “ba” результатом должна быть строка ”a*ba*b*ba*”.

```
char * replace(char * source, char * target, char * subst);
```

Написать функцию change_base, переводящую текстовое представление num неотрицательного числа из системы счисления b1 в систему b2. Цифры, большие 9, обозначить латинскими буквами A, B, C,... Основания систем счисления не больше 30. Например, при num = “1001011”, b1 = 2, b2 = 16 результатом должна быть строка “4B”.

```
char * change_base(char * num, int b1, int b2);
```

Задача на засыпку: то же, но без ограничения на размер числа.

Задача на засыпку:
Написать рекурсивную функцию det вычисления определителя матрицы A размера N*N.

```
#define N 1000
double next_perm(double A[][]);
```

Задача на засыпку:
Написать функцию list_circular проверки того, является ли данный односвязный список циклическим. Функция должна работать за время, пропорциональное количеству элементов в списке, и использовать память, размер которой не зависит от количества элементов. Функция не должна изменять входной список.

```
struct SList
{
    int value;
    struct SList * next;
}
#define bool unsigned int
bool list_circular(SList * list)
```

Ориентированный граф задан матрицей смежности M размера N*N. Найти все вершины, в которые есть путь из вершины с номером b. Результат является логическим вектором длины N, где i-ый элемент истинен, если i-ая вершина достижима их k-ой.

```
#define N 1000
#define bool unsigned int
bool * reachable_nodes(bool M[][], int k);
```