Набор задач `task2`

Пробельной последовательностью (ПП) называется последовательность одного или нескольких символов пробела, табуляции, или перевода строки.

Стандартный входной поток обозначается `stdin`. Стандартный выходной поток — `stdout`. Стандартный выходной поток ошибок — `stderr`.

Проверка правильности программ проводится полуавтоматическим способом, поэтому необходимо строго соблюдать входной и выходной формат, описанный в задании. Любую отладочную информацию можно выводить на поток `stderr` — он игнорируется.

- `task2.1`

  С потока `stdin` задается вещественное число ε. Далее, пока не достигнут конец файла (`ctrl-D` на клавиатуре), вводится очередное вещественное число x. Для каждого введенного x вычислить и вывести в `stdout` значение $\sqrt{x}$. Для вычисления квадратного корня использовать итеративную формулу (метод Ньютона):

  $$x_0 = 1,\ x_{i+1} = \frac{1}{2}\left(x_i + \frac{x}{x_i}\right),\ i = 0, 1, 2, \ldots$$

  Вычисления указанным способом продолжать, пока $|x_i - x_{i+1}| \geq \varepsilon$. Ответом считается число x_{i+1}. Каждый ответ размещается на отдельной строке. Для печати использовать `printf()` со спецификацией форматного преобразования `%.10g`. Для хранения и операций с вещественными числами использовать тип `double`. Не использовать никаких стандартных математических функций языка Си.

- `task2.2.1`

  В `stdin` задана последовательность вещественных чисел:

  $$x, a_n, a_{n-1}, \ldots, a_0,$$

  разделенных ПП. Конец файла — признак конца последовательности. Вывести в `stdout` вычисленное по схеме Горнера значение многочлена с коэффициентами $a_0, \ldots, a_n$ в точке x (a_0 — свободный член, a_n — коэффициент при x^n). Не использовать массивы и указатели.

- `task2.2.2`

  Вариант задачи `task2.2.1`. Кроме значения многочлена, вывести на следующей строке значение его производной в точке x.

- `task2.2.3`

  Вариант задачи `task2.2.2`. Кроме значения многочлена и производной, вывести на следующей строке значение определенного интеграла от этого многочлена с пределами интегрирования от 0 до x. Для этого изменить входную спецификацию:

  $$n, x, a_n, a_{n-1}, \ldots, a_0$$

  (в начале последовательности вводится неотрицательное число n – степень многочлена).

- `task2.3`

  Описать итеративную и рекурсивную версии функции для вычисления i-го члена последовательности Фибоначчи:

  $$F_0 = 0,\ F_1 = 1,\ F_i = F_{i-2} + F_{i-1},\ i = 0, 1, 2, 3, \ldots$$

  Написать программу, которая вводит с `stdin` последовательность целых чисел до конца файла. Как только введено очередное число i, для него напечатать на отдельной строке i-е число Фибоначчи, вычисленное с помощью итеративной функции, затем на другой строке это же число, вычисленное с помощью рекурсивной функции. Пронаблюдать разницу во времени вычисления.

- `task2.4`

  Описать функцию с прототипом

  ```
  double str2double( char str[] );
  ```

  которая преобразует строку в вещественное число типа `double`. Вещественное число задается по правилам языка Си (см. синтаксическую. диаграмму понятия ≪вещественная константа≫), исключая суффикс `L` (т.е. `long double`). Не использовать стандартные функции.

  Написать программу, которая вводит, из `stdin`, пока не конец файла, корректные записи вещественных чисел, разделенные ПП. Для чтения записей чисел использовать спецификацию `%s` функции `scanf`. Каждую введенную запись программа переводит с помощью вашей функции `str2double()` в число типа `double` и выводит его число в `stdout`, используя спецификацию формата `%.10g`. Каждое число выводить в отдельной строке.

- `task2.5`

  Описать тип ≪список слов≫ на языке Си. Список реализовать как цепочку динамических объектов, создаваемых стандартными функциями `malloc()` или `calloc()`. Написать программу, которая вводит строку, состоящую из непустых слов, разделенных последовательностями пробелов и табуляций, и строит список из этих слов. Слова размещать в памяти как динамические объекты. Далее из списка исключить все слова, совпадающие с последним (само последнее слово остается). Список может быть пустым, если в строке не было слов, тогда результат по определению пустой. Преобразованный список слов напечатать в `stdout`, отделяя слова одним пробелом, в конце списка — перевод строки.

- `task2.6`

  Описать тип ≪дерево поиска≫ с неотрицательными целыми ключами. Написать программу, которая вводит, пока не конец файла, из `stdin` элементы трех видов (разделенные ПП):

  ```
  +<неотрицательное целое число>
  ```

  ```
  -<неотрицательное целое число>
  ```

  ```
  ?<неотрицательное целое число>
  ```

Число с предшествующим плюсом добавляется в дерево поиска, если его в нем нет; если числу предшествует минус, оно удаляется из дерева поиска, если оно в нем есть. Если перед числом стоит знак вопроса, то оно печатается в `stdout` с новой строки, затем пробел, затем слово `yes` или `no` (соблюдайте регистр букв) в зависимости от того, присутствует ли это число в дереве поиска, или нет.

- `task2.7`

Ниже приведена программа-калькулятор, которая вычисляет выражения, состоящие из цифр `'0'-'9'`, операций сложения, умножения и скобок. Приоритет умножения выше, чем у сложения. Операции левоассоциативны. Программа действует методом рекурсивного спуска, и построена на основе следующей грамматики:

`<выражение>::= <слагаемое> {+ <слагаемое>}`

`<слагаемое>::= <множитель> {* <множитель>}`

`<множитель>::= 0|1|2|3|4|5|6|7|8|9| (<выражение>)`

Дополнить калькулятор операциями вычитания (обозначается '-', левоассоциативна, приоритет как у сложения), деления с отбрасыванием остатка (обозначается '/', левоассоциативна, приоритет как у умножения), возведения в степень (обозначается ^, правоассоциативна, т.е. $2\hat{}1\hat{}2 = 2$, а не 4, приоритет самый высокий, возведение в отрицательную степень — ошибка). Для этого построить подходящую грамматику выражений, а по ней — анализатор, действующий методом рекурсивного спуска.

```
/*  Вычисление значения выражения, содержащего цифры '0'-'9', знаки
 *  операций '+', '*' и скобки '(', ')'. ( Предполагается, что коды цифр
 *  упорядочены по возрастанию цифр и справделиво равенство '9'-'0'==9 как,
 *  например, в ASCII кодировке)
 */

#include<stdio.h>
#include<setjmp.h>

jmp_buf begin;  /* точка начала диалога с пользователем */

char curlex;  /* текущая лексема */

void getlex(void); /* выделяет из входного потока очередную лексему */

int expr(void); /* распознает выражение и вычисляет его значение */

int add(void); /* распознает слагаемое и вычисляет его значение */

int mult(void); /* распознает множитель и вычисляет его значение */

void error(); /* сообщает об ошибке в выражении и передает управление
 в начало функции main  (точка begin) */


main()
{
  int result;
  setjmp(begin);
  printf("==>");
```

```
  getlex();
  result=expr();
  if (curlex != '\n') error();
  printf("\n%d\n",result);
  return 0;
}

void getlex()
{
  while ( (curlex=getchar()) == ' ');
}

void error(void)
{
  printf("\nОШИБКА!\n");
  while(getchar()!='\n');
  longjmp(begin,1);
}

int expr()
{
  int e=add();
  while (curlex == '+')
    { getlex(); e+=add();}
  return e;
}

int add()
{
  int a=mult();
  while (curlex == '*')
    { getlex(); a*=mult();}
  return a;
}

int mult()
{
  int m;
  switch(curlex){
    case '0': case '1': case '2': case '3': case '4': case '5':
    case '6': case '7': case '8': case '9': m=curlex-'0'; break;
    case '(': getlex(); m=expr();
      if (curlex == ')') break;
      /* иначе ошибка - нет закрывающей скобки */
    default : error();
  }
  getlex();
  return m;
}

/* Сравните прототипы (в начале программы) и заголовки функций
 * (в соответствующих определениях) error() и getlex() : информацию
 * о параметрах (в данном случае пустые списки параметров - void) можно
 * опускать в заголовке, если она есть в объявлении прототипа (как в случае
 * с getlex)  или не указывать в прототипе, а указать только в заголовке
 * (как в случае с error). Тип обех функций - "функция, не возвращающая
 * значения с пустым списком аргументов"
 */
```