Набор задач `task2`

1. Со стандартного потока ввода задается вещественное число ε. Далее, пока не достигнут конец файла (`ctrl-D` на клавиатуре), вводится очередное вещественное число x. Для каждого введенного x вычисляется и выводится в стандартный выходной поток значение $\sqrt{x}$. Для вычисления квадратного корня используется итеративная формула (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 продолжаются, пока $|x_i - x_{i+1}| \geq \varepsilon$. Ответом является число x_{i+1}. Каждый ответ должен размещаться на отдельной строке. Для печати использовать `printf()` со спецификацией форматного преобразования `double`. Не используйте никаких стандартных математических функций языка Си.

2. В стандартном входном потоке задана последовательность вещественных чисел: $x, a_n, a_{n-1}, \ldots, a_0$. Конец файла — признак конца последовательности. Вывести в стандартный выходной поток вычисленное по схеме Горнера значение многочлена с коэффициентами $a_0, \ldots, a_n$ в точке x (a_0 — свободный член, a_n — коффициент при x^n). Не использовать массивы и указатели. Усложненный вариант задачи: кроме значения многочлена, вывести значение его производной в точке x и определенного интеграла от заданного многочлена с пределами интегрирования от 0 до x.

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 со стандартного входного потока последовательность целых чисел до конца файла. Как только введено очередное число i, для него вычисляется и сразу печатается i-е число Фибоначчи с помощью итеративной функции, затем — это же число, вычисленное с помощью рекурсивной функции.

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. синт. диаграмму понятия "вещественная константа"), исключая суффикс `L` (т.е. `long double`). Не использовать стандартные функции.

   Написать программу, которая вводит, пока не конец файла, строки из стандартного входного потока (каждая строка содержит корректную запись веществ. числа), переводит строку с помощью вашей функции `str2double()` в число типа `double` и выводит это число в стандартный поток вывода, используя спецификацию формата `%.10g`. Каждое число выводится в отдельной строке. Для чтения строк используйте спецификацию `%s` функции `scanf`.

5. Опишите тип "список слов"на языке Си. Список реализуйте как цепочку динамических объектов, создаваемых стандартными функциями `malloc()`

или `calloc()`. Напишите программу, которая вводит строку, состоящую из непустых слов (разделенных одним или несколькими пробелами или табуляциями), и строит список из этих слов. Слова размещайте в памяти как динамические объекты. Далее из списка исключаются все слова, совпадающие с последним (само последнее остается). Список может быть пустым, если в строке не было слов. Преобразованный список слов напечатать в стандартный выходной поток, отделяя слова одним пробелом, в конце — символ новой строки.

6. Опишите тип "дерево поиска"с целыми ключами. Напишите программу, которая вводит, пока не конец файла, из стандартного входного потока элементы трех видов:

   `+<неотрицат. целое число>`

   `-<неотрицат. целое число>`

   `?<неотрицат. целое число>`

   Элементы отделяются друг от друга одним или несколькими разделителями — пробелами , табуляциями, символами новой строки. Число с предшествующим плюсом добавляется в дерево поиска, если его там еще нет; если числу предшествует минус, то это (неотрицат.) число удаляется из дерева поиска (если оно в нем было). Если перед числом вопрос, то оно печатается в выходной поток в отдельной строке вместе со словом `Yes` или `No` в зависимости от того, присутствует ли это число в дереве поиска.

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 грамматики:

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

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

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

   Обогатите 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. 2^1^2=2, а не 4, приоритет самый высокий, возведение в отрицательную степень — ошибка). Для этого постройте подходящую грамматику выражений и 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" */
```