1. オブジェクト指向を用いると，ファイル入出力やデータ構造（**コレクション**）等に関連する処理をまとめ，クラスという形で部品化することができる。このようなクラスライブラリ（C#では.NET Framework）を使うには，**例外処理**や**ジェネリクス**といった機能の概要も知る必要がある。
　下記のプログラムは，キーボードから正の整数を読み込んでスタックに保存した後，ユーザが指定したファイルに逆順に書き出すものである。空欄を適切に埋めて実行し，動作を確認せよ。

```
using System;
using System.Collections.Generic;  // コレクションライブラリ
using System.IO;  // 入出力ライブラリ

class Program {
    public static void Main() {
        // Stack はジェネリッククラスであり，<>で渡した型に対応するクラスが生成される
        // この場合（<int>）は，int 型を要素とする「int 型用の Stack クラス」が生成される

        Stack<int> stack =

        while (true) {
            string line = Console.ReadLine();
            if (line == null) break;

            // 例外処理：try ブロックの処理の中で，エラー等の例外（exception）が発生すると，
            // それ以降の処理は中断され，発生した例外に対応した catch ブロックに実行が移る
            try {
                int data = int.Parse(line);  // Parse は FormatException を発生し得る
                if (data < 0) break;
                stack.Push(data);  // スタックにデータをプッシュする
            }
            catch (FormatException e) { // Parse が失敗した場合はこの例外処理を実行する
                continue;
            }
        }

        Console.Write("File name: ");
        string fname = Console.ReadLine();
        FileStream fstream;

              {
            // ファイルを書き込みモード（FileMode.Create）で開く（失敗なら入出力例外が発生）

            fstream = new FileStream
        }
                  (IOException e) { // ファイルのオープンに失敗した場合

            Console.WriteLine("Coundn't open.");
            return;
        }
        // StreamWriter は文字列単位での書き出し（Write, WriteLine メソッド）を提供する

        StreamWriter writer =

        while (stack.Count > 0) {
            // スタックからデータをポップし，データをストリームに書き出す

            int data =

            writer.
        }
        writer.Close();  // ストリームを閉じる
    }
}
```

2. 下記のプログラムは，ユーザが選択した英文（欧文）のテキストファイルを開き，それに含まれる各単語の出現回数を数えて表示するものである。C#は**型推論**をサポートしており，変数宣言でデータ型の代わりに var と記述されている場合，初期値等から自動的に型を決定する。プログラムを入力して実行させ，動作を確認せよ。また，それぞれの var は実際にはどのような型になるか考えよ（必要なら調べよ）。

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text.RegularExpressions;  // 正規表現ライブラリ
using System.Windows.Forms;

class Program {
    [STAThread]  // GUI を使うプログラムで必要
    public static void Main() {
        // 単語（文字列）をキー（Key）として出現回数を値（Value）とする Dictionary クラスの生成
        //  （Dictionary の実装はハッシュテーブルなので，追加・削除の計算量はほぼ O(1)である）
        var seen = new Dictionary<string, int>();

        // コンソールプログラムで無理やりファイル選択ダイアログを開く（公式な動作保証は不明）
        var dialog = new OpenFileDialog();
        var result = dialog.ShowDialog();
        if (result != DialogResult.OK) return;
        var fname = dialog.FileName;

        // 読み込みモード（FileMode.Open）でファイルを開く
        FileStream fstream;  // 初期値がない場合は var は使えない
        try {
            fstream = new FileStream(fname, FileMode.Open);
        }
        catch (IOException) {
            Console.WriteLine("Couldn't open.");
            return;
        }
        var reader = new StreamReader(fstream);  // ReadLine メソッドを提供

        for (;;) {
            var line = reader.ReadLine();
            if (line == null) break;

            // 正規表現（Regex クラス）用い，文字列 line を単語の配列に分割（Split）する
            // その際，英数字以外（¥W）の 1 文字以上の連続（+）を分割の区切りとする
            // 文字列リテラルに@を付けると通常のエスケープシーケンス（¥n 改行等）が無効になる
            var words = Regex.Split(line, @"¥W+");
            foreach (var w in words) {
                // 切り出した単語をすべて小文字に変換する
                var key = w.ToLower();

                // Dictionary では配列のように要素に対して mame[key] という形式でアクセスできる
                // このように，配列と類似した記法で整数以外を添字に使えるものを連想配列という
                if (seen.ContainsKey(key))
                    seen[key]++;    // 登録済みの単語ならば出現回数を 1 増やす
                else
                    seen[key] = 1;  // 新しい要素を登録するには代入が使える
            }
        }
        reader.Close();

        // 全要素を表示する
        foreach (var pair in seen) {
            Console.WriteLine("{0,5:d} {1}", pair.Value, pair.Key);
        }
        // Console.ReadLine();
    }
}
```