# Zadanie: KUC
# Kucharz

Olimpiada Informatyczna

**XXIV OI, etap III, dzień drugi. Plik źródłowy** `kuc.*` **Dostępna pamięć: 8 MB.** *12.04.2017*

Kucharz Bajtazar pracuje w restauracji i ma $n$ zamówień czekających na przygotowanie. Każde zamówienie zapisane jest na karteczce, a wszystkie karteczki są nadziane na szpikulec. Bajtazar musi wykonywać zamówienia od góry, ponieważ może ściągać karteczki tylko z góry szpikulca. Przyrządzanie dań jest czasochłonne i kucharz chciałby wykonać wszystkie zamówienia możliwie jak najszybciej. Nie musi jednak wykonywać ich pojedynczo. Załóżmy, że na szpikulcu zostało $k$ karteczek. Oto co może zrobić:

- Wziąć jedno zamówienie z góry i przyrządzić je w czasie `jedno`($k$).
- Jeśli $k > 1$, to może wziąć dwa zamówienia z góry i przyrządzić je w czasie `dwa`($k$).
- Jeśli $k > 1$, to może wziąć $\lfloor k/2 \rfloor$ zamówień z góry i wykonać je w czasie `polowa`($k$).

Dodatkowo, Bajtazar ma pewien poziom energii, początkowo równy $e$. Operacje trzeciego rodzaju, tzn. te biorące połowę zamówień, bardzo go męczą i obniżają jego poziom energii o 1. Jest jednak mistrzem w operacjach pierwszego rodzaju (czyli w obsługiwaniu jednego zamówienia) i każda taka operacja podnosi jego poziom energii o 1. Poziom energii kucharza nie może nigdy spaść poniżej 0. Bajtazara nie interesuje jego końcowy poziom energii; chciałby jedynie jak najszybciej przygotować wszystkie zamówienia.

Napisz program komunikujący się z biblioteką dającą wszystkie potrzebne informacje na temat stanu Bajtazara i jego kuchni, który znajdzie najmniejszy możliwy czas, w jakim Bajtazar będzie w stanie przygotować wszystkie zamówienia. W tym zadaniu zwróć szczególną uwagę na limit dostępnej pamięci.

## Komunikacja

Aby użyć biblioteki, należy wpisać na początku programu:

- **C/C++**: `#include "ckuclib.h"`
- **Pascal**: `uses pkuclib;`

Biblioteka udostępnia następujące funkcje i procedury:

- `dajn`, `daje`
  Pierwsza funkcja daje w wyniku liczbę całkowitą $n$, oznaczającą liczbę zamówień do przygotowania, zaś druga – liczbę całkowitą $e$, oznaczającą początkowy poziom energii Bajtazara.
  - **C/C++:**
    ```
    int dajn();
    int daje();
    ```
  - **Pascal:**
    ```
    function dajn: LongInt;
    function daje: LongInt;
    ```
- `jedno`($k$), `dwa`($k$), `polowa`($k$)
  Funkcje dają w wyniku czas przyrządzania zamówień, gdy na szpikulcu jest aktualnie $k$ karteczek, a Bajtazar postanowi zrealizować, odpowiednio, jedno, dwa lub $\lfloor k/2 \rfloor$ zamówień. Dla $k = 1$ wartości funkcji `dwa`($k$) i `polowa`($k$) są nieistotne. Czas jest liczbą całkowitą z przedziału od 1 do $10^7$.
  - **C/C++:**
    ```
    int jedno(int k);
    int dwa(int k);
    int polowa(int k);
    ```
  - **Pascal:**
    ```
    function jedno(k: LongInt): LongInt;
    function dwa(k: LongInt): LongInt;
    function polowa(k: LongInt): LongInt;
    ```
- `odpowiedz`($wynik$)
  Odpowiada bibliotece, że $wynik$ to najmniejszy możliwy czas, w którym Bajtazar może przygotować wszystkie zamówienia. Wywołanie tej funkcji **kończy działanie Twojego programu**.
  - **C/C++:**
    ```
    void odpowiedz(int wynik);
    ```
  - **Pascal:**
    ```
    procedure odpowiedz(wynik: LongInt);
    ```

Twój program **nie może** czytać żadnych danych (ani ze standardowego wejścia, ani z plików). **Nie może** również nic wypisywać do plików ani na standardowe wyjście. Może pisać na standardowe wyjście diagnostyczne (`stderr`) – pamiętaj jednak, że zużywa to cenny czas. Bibliotekę można pytać wielokrotnie o dane wartości.

# Ocenianie

Zestaw testów dzieli się na następujące podzadania. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

| Podzadanie | Warunki | Liczba punktów |
|---|---|---|
| 1 | $n, e \leq 1000$ | 12 |
| 2 | $n, e \leq 50\,000$ | 8 |
| 3 | $n, e \leq 1\,000\,000$ | 80 |

# Przykładowy przebieg programu

| C/C++ | Pascal | Wynik |
|---|---|---|
| `n = dajn();` | `n := dajn;` | 3 |
| `e = daje();` | `e := daje;` | 1 |
| `p[3] = polowa(3);` | `p[3] := polowa(3);` | 1 |
| `p[2] = polowa(2);` | `p[2] := polowa(2);` | 4 |
| `j[2] = jedno(2);` | `j[2] := jedno(2);` | 2 |
| `j[1] = jedno(1);` | `j[1] := jedno(1);` | 5 |
| `d[2] = dwa(2);` | `d[2] := dwa(2);` | 6 |
| `odpowiedz(7);` | `odpowiedz(7);` | — |

Powyższy przebieg programu jest poprawny pod względem formalnym, choć niekoniecznie daje poprawny wynik. Z podanych informacji wnioskujemy, że wykonanie operacji `polowa` i `dwa` (o koszcie $p[3] + d[2] = 7$) jest lepsze od wykonania jednej operacji `polowa` i dwóch operacji `jedno` (o koszcie $p[3] + j[2] + j[1] = 8$). Wiemy też, że nie opłaca się wykonywać operacji `jedno` przy trzech zamówieniach (bo $p[3] = 1$, a $j[3] \geq 1$). Wykonanie dwóch operacji `polowa` nie jest możliwe ze względu na to, że $e = 1$. Ale bez znajomości kosztu operacji `dwa(3)` nie możemy stwierdzić, czy wykonanie kolejno operacji `dwa` i `jedno` nie dałoby lepszego wyniku.

# Eksperymenty

W katalogu `dlazaw` dostępna jest przykładowa biblioteka, która pozwoli Ci przetestować poprawność formalną rozwiązania. Biblioteka wczytuje informacje ze standardowego wejścia w następującym formacie:

- w pierwszym wierszu dwie liczby całkowite $n$ i $e$;
- w drugim wierszu $n$ liczb całkowitych z przedziału $[1, 10^7]$ – wartości funkcji `jedno` dla kolejnych liczb $k = 1, \ldots, n$;
- w trzecim i czwartym wierszu wartości dla funkcji `dwa` i `polowa` (w takim samym formacie); jako pierwsza musi pojawić się wartość dla $k = 1$, ale jest ona nieistotna.

Przykładowe wejście dla biblioteki znajduje się w pliku `kuc0.in`. Po wywołaniu procedury `odpowiedz`, biblioteka wypisuje na standardowe wyjście informację o udzielonej odpowiedzi.

W tym samym katalogu znajdują się przykładowe rozwiązania `kuc.c`, `kuc.cpp` i `kuc.pas` korzystające z biblioteki. Rozwiązania te nie są poprawne i zawsze wykonują operację `polowa`, poza ostatnim wywołaniem, w którym wykonują operację `jedno`.

Do kompilacji rozwiązania wraz z biblioteką służą polecenia:

- **C**: `gcc -O2 -static ckuclib.c kuc.c -lm -std=gnu99`
- **C++**: `g++ -O2 -static ckuclib.c kuc.cpp -lm -std=c++11`
- **Pascal**: `ppc386 -O2 -XS -Xt kuc.pas`

Plik z rozwiązaniem i biblioteka powinny znajdować się w tym samym katalogu.