MATEMATICKÁ OLYMPIÁDA NA STREDNÝCH ŠKOLÁCH

52. ročník, školský rok 2002/2003

Zadania úloh 3. kola kategórie P

1. súťažný deň

Na riešenie úloh máte 4.5 hodiny čistého času. Riešenie každého príkladu musí obsahovať (pokiaľ nie je v zadaní uvedené ináč):

- **Popis riešenia,** to znamená slovný popis použitého algoritmu, argumenty zdôvodňujúce jeho správnosť (prípadne dôkaz správnosti algoritmu), diskusiu o efektivite vášho riešenia (časová a pamäťová zložitosť). Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného zápisu algoritmu (do programu).
- **Program.** V úlohách **P-III-1** a **P-III-2** treba uviesť dostatočne podrobný zápis algoritmu, najlepšie v tvare zdrojového textu najdôležitejších častí programu v jazyku Pascal alebo C. Zo zápisu môžete vynechať jednoduché operácie ako vstupy, výstupy, implementáciu jednoduchých matematických vzťahov a pod. V úlohe **P-III-3** uveďte namiesto toho svoje riešenie ako reverzibilnú procedúru.

Hodnotí se nielen správnosť programu, ale tiež kvalita popisu riešenia a efektivita zvoleného algoritmu.

P-III-1

V hračkárstve Drobec a otec prebehla veľká súťaž "O najkrajšiu hračku". Úlohou detí bolo nakresliť obrázok svojej najobľúbenejšej hračky. Po skončení súťaže sa konala výstavka a deti, ktoré nakreslili najkrajšie obrázky, dostali od hračkárstva nejakú hračku. Ale ako asi viete, nie každému dieťaťu sa páčia všetky hračky, a tak už pred ukončením súťaže mal každý výtvarník vyhliadnutú odmenu, ktorú by chcel za svoj obrázok dostať. Tú, a žiadnu inú. Po vyhlásení výsledkov dávali deti svoj názor najavo dosť nahlas. Maminky ukričaných potomkov sa rozhodli, že deti si medzi sebou hračky povymieňajú tak, aby pokiaľ možno čo najviac detí bolo spokojných so svojou výhrou. Situáciu komplikuje skutočnosť, že zapojiť sa do vymieňania sú ochotné iba tie deti, ktoré nakoniec dostanú tú hračku, po ktorej túžia. S tak náročnou úlohou si mamičky nevedli poradiť, a tak poprosili Vás, aby ste napísali program, ktorý úlohu vyrieši.

Váš program dostane na vstupe zadaný počet odmenených detí N a ďalej pre každé dieťa číslo hračky, ktorú dostalo a číslo hračky, ktorú by chcelo dostať (pretože hračiek je rovnako ako detí, očíslujeme si ich pre jednoduchosť od 1 po N). Na výstup program vypíše najväčšiu skupinu detí takú, že ak si deti v skupine medzi sebou vhodným spôsobom vymenia hračky, budú všetky spokojné.

Príklad:

Predpokladajme, že prvé dieťa má hračku 2 a chce hračku 4, druhé dieťa má hračku 3 a aj chce hračku 3, tretie dieťa má hračku 5 a chce hračku 2, štvrté dieťa má 4 a chce 5 a piate dieťa má 1 a chce 2. Najväčšia skupina detí, ktoré môžu výmenou dostať svoje vytúžené hračky, obsahuje deti 1, 2, 3, a 4.

P-III-2

Knihovníčka Milka opäť potrebuje objednať skriňu do svojej knižnice a keďže sa jej vaša pomoc osvedčila, opäť sa na Vás obrátila, aby ste jej pomohli spočítať optimálne rozmery novej skrine. Nová skriňa má mať P poličiek a Milka by do nej chcela umiestniť N kníh. Každá kniha má priradený jednoznačný číselný kód a tieto kódy určujú poradie kníh v skrini. Kniha s menším kódom sa má nachádzať na rovnakej alebo vyššie umiestnenej poličke ako kniha s väčším kódom. Na každej poličke majú byť knihy s menšími kódmi umiestené naľavo od kníh s väčšími kódmi.

Vstupom pre váš program budú čísla P a N a postupnosť N čísel h_i, $1 \leq i \leq N$, kde h_i je hrúbka i-tej knihy (v poradí podľa kódov od najmejšieho po najväčší). Môžete predpokladať, že hrúbka h_i i-tej knihy je v rozmedzí od 1 mm do 50 mm. Každá kniha ma takú výšku, že sa dá bez problémov umiestniť do ľubovoľnej z plánovaných P poličiek. Váš program by mal zo zadaných údajov spočítať nasledujúce údaje:

- Šírku skrine – označme ju s.
- Rozmiestenie kníh do skrine s vypočítanými parametrami.

Rozmiestnenie kníh, ktoré váš program nájde, musí z pochopiteľných dôvodov spĺňať nasledujúce požiadavky:

- Súčet hrúbok kníh umiestnených do jednej poličky je najviac s.
- Šírka skrine s je najmenšia možná.

Príklad:

Predpokladajme, že nová skriňa má mať 3 poličky a má v nej byť uložených 6 kníh, ktorých hrúbky v poradí podľa ich kódu sú nasledujúce:

15 mm, 20 mm, 7 mm, 6 mm, 2 mm a 4 mm. Minimálna možná šírka skrine v tomto prípade je 20 mm. Na prvú poličku se dá iba prvá kniha, na druhú iba druhá kniha a zvyšné knihy sa umiestnia na poslednú tretiu poličku.

P-III-3

Študijný text je rovnaký ako v predchádzajúcich kolách, nájdete ho za zadaním súťažnej úlohy.

Súťažná úloha

Vašou úlohou je napísať *reverzibilnú* procedúru `Add(var n:word; var A,B: array [0..n-1] of bit)`, ktorá bude slúžiť na sčítanie n-bitových čísel. Na vstupe dostane v poliach A, B dve n-bitové čísla zapísané po bitoch v dvojkovej sústave (pričom $A[0]$ je najnižší bit). Po skončení procedúry by v poli A mal byť uložený súčet týchto dvoch čísel. Môžete predpokladať, že vstup bude taký, aby nenastalo pretečenie, t.j. súčet bude menší ako 2^n. Základným kritériom hodnotenia bude **pamäťová zložitosť** vašej procedúry.

Študijný text

Pri hľadaní úspornejších polovodičových technológii sa zistilo, že najviac energie sa spotrebúva pri mazaní informácií. Teda optimálne sú tie výpočty, pri ktorých sa žiadne informácie nestrácajú. Takýmto výpočtom sa hovorí *reverzibilné*, lebo vďaka tejto vlastnosti môžu prebiehať oboma smermi – dá sa určiť nielen zo vstupu výstup, ale aj z výstupu vstup. Aj my sa teraz vydáme do tohoto zvláštneho symetrického sveta a preskúmame, ako sa programuje „ekologicky“.

Začneme tým najjednoduchším, čo v klasických programovacích jazykoch máme, a to priraďovacím príkazom. Tu si nič také, žiaľ, nemôžeme dovoliť, lebo by sme stratili pôvodný obsah premennej, do ktorej priraďujeme! Namiesto toho zavedieme niekoľko príkazov, ktoré budú premennú modi kovať vratne:

- `premenná += hodnota` – pripočíta hodnotu k premennej
- `premenná -= hodnota` – odpočíta hodnotu od premennej
- `premenná ^= hodnota` – prixoruje hodnotu k premennej
- `premenná =:= premenná` – vymení obsah dvoch premenných

Operácia **xor** je bitová operácia, ktorá má pre dve jednobitové čísla výsledok 1 práve vtedy, keď sú rôzne. Teda (0 xor 0) = (1 xor 1) = 0

a (0 xor 1) = (1 xor 0) = 1. Viacbitové čísla sa xorujú po bitoch – i-tý bit výsledku je i-tý bit prvého čísla **xor** i-tý bit druhého čísla. Napr. 5 xor 14 = $(0101)_2$ xor $(1110)_2$ = $(1011)_2$ = 11. Operácia **xor** má veľa užitočných vlastností, okrem iného $(x \text{ xor } y) = (y \text{ xor } x)$, $(x \text{ xor } x) = 0$, $(x \text{ xor } 0) = x$ a $((x \text{ xor } y) \text{ xor } z) = (x \text{ xor } (y \text{ xor } z))$. Podobne sa dajú zaviesť aj bitový **and** a **or**: je (0 and 0) = (0 and 1) = (1 and 0) = 0, (1 and 1) = 1, (0 or 0) = 0, (0 or 1) = (1 or 0) = (1 or 1) = 1. Tieto operácie nás až tak nebudú zaujímať, lebo nie sú reverzibilné.

Aby sme sa vyhli problémom s pretečením (čo je potom inverzná operácia?), dohodneme sa, že budeme počítať len s nezápornými celými číslami od 0 do `maxword`. Takýmto číslam budeme odteraz hovoriť *prirodzené*. Všetky operácie budeme robiť modulo (`maxword`+1), čiže výsledkom každej z operácií na prirodzených číslach bude opäť prirodzené číslo. Navyše príkaz `-=` bude naozaj inverzný k `+=` a naopak. Príkazy `^=` a `=:=` sú zjavne inverzné samy k sebe.

Čo všetko ale môže byť `hodnota`? Iste to môže byť ľubovoľná konštanta. Takisto to môže byť ľubovoľná premenná, samozrejme okrem tej, do ktorej priraďujeme. Inak by sme mohli napísať napr. „`a -= a`“, čo zjavne nie je reverzibilné. Ešte by sme mali povoliť základné aritmetické operácie – tie samy nemusia byť reverzibilné, stačí, keď reverzibilne spracujeme ich výsledok. Každý zložitejší výraz potom môžeme prepísať na výrazy s jednou operáciou, napr. „`x ^= (a*b)+(c*d)`“rozpíšeme takto:

```
t1 += a*b;  t2 += c*d;  x ^= t1+t2;  t2 -= c*d;  t1 -= a*b;
```

Pritom `t1` a `t2` sú pomocné premenné, ktoré sú na počiatku výpočtu nulové a po dopočítaní výrazu sa opäť k nulovým vrátia, takže ich môžeme znovu použiť. Podobne sa dá rozpísať do reverzibilného tvaru výpočet ľubovoľného výrazu, takže odteraz môžeme používať aj zložené výrazy (bez toho, aby sme ich museli rozpisovať).

Trik s odpočítavaním medzivýsledkov a spúšťaním častí programu odzadu sa nám ešte môže hodiť. Zade nujme si teda, že `undo prikaz` znamená spustiť príkaz odzadu a že `wrap prikaz1 on prikaz2` najskôr vykoná `prikaz1`, potom `prikaz2` a nakoniec `undo prikaz1`. Náš príklad s postupným výpočtom výrazu by sme teda mohli prepísať nasledovne:

```
wrap begin  t1 += a*b;  t2 += c*d;  end on  x ^= t1+t2;
```

Podmienené príkazy `if-then-else` môžeme používať, ak zaručíme, že po vykonaní podmieneného príkazu bude pravdivosť podmienky rovnaká ako pred jeho vykonaním (napríklad preto, že žiadnu z premenných, ktoré sú v podmienke, v podmienenej časti programu nemeníme). Potom totiž vieme aj pri vykonávaní výpočtu odzadu rozhodnúť, ktorou vetvou sa má výpočet vydať.

Ťažšie to bude u cyklov. Tam si nevystačíme s nemeniacou sa pod-

mienkou – to by cyklus buď neprebehol ani raz, alebo sa opakoval do nekonečna. My budeme používať cykly typu `for`. Tie budú reverzibilné, ak vnútri cyklu nikde nemeníme riadiacu premennú cyklu, ani jej hranice. Toto nie je až také veľké obmedzenie – nesmie sa to robiť ani v niektorých obyčajných programovacích jazykoch. Navyše aby nás netrápilo, čo bolo v riadiacej premennej pred začiatkom cyklu a čo je v nej po jeho skončení, dohodneme sa, že príkaz `for` si túto premennú sám vytvorí a na konci ju zase zruší.

Príkaz `goto` pre istotu zakážeme úplne.

Procedúry môžu fungovať reverzibilne, ale musíme sa vyhnúť kopírovaniu parametrov a výsledkov. Všetky premenné preto budeme procedúre odovzdávať odkazom (v Pascale `var`, v C `*`). Lokálne premenné procedúry budú pri jej spustení nulové a procedúra ich musí pred skončením opäť uviesť do tohoto stavu. Rekurzia funguje bez problémov.

Teraz už máme všetko pripravené na to, aby sme vybudovali reverzibilný programovací jazyk. Bude príbuzný Pascalu a bude vyzerať takto:

Dátové typy. K dispozícii máme typy `word` (nezáporné celé číslo), `bit` (jednobitové číslo, t.j. 0 alebo 1, používa sa aj pre pravdivostné hodnoty ako pascalovský `boolean`) a pole `array[x..y] of typ` (`x` a `y` udávajú rozmedzie indexov a okrem čísel to môžu byť aj výrazy, ktorých hodnota sa počas existencie poľa nezmení). Prvky polí môžu byť aj polia, takto získame viacrozmerné polia. Svoj vlastný typ si môžete zaviesť deklaráciou `type identifikator = typ`, napr.:

```
type boolean = bit;
type screen = array[0..199] of array[0..319] of bit;
```

Identifikátory slúžia na pomenovanie typov, premenných a procedúr a sú to ľubovoľné reťazce písmen, číslic a znakov '_', ktoré nezačínajú číslicou a nezhodujú sa s niektorým z kľúčových slov jazyka. Malé a veľké písmená sa nerozlišujú.

Procedúry sa deklarujú konštrukciou:

```
procedure identifikator ( parametre );
  (deklarácie lokálnych typov, premenných a procedúr)
begin
  (príkazy oddelené bodkočiarkami)
end;
```

Parametre procedúry majú syntax `var meno : typ`, kde `meno` je identifikátor, ktorým sa na parameter odkazujeme vnútri procedúry. Ak je parametrov viac, oddeľujú sa pri deklarácii procedúry bodkočiarkami. Ak sú parametre rovnakého typu, môžeme zápis skracovať, napr. `procedure X(var m,n : word; var A:array[1..n] of bit);` Všetky objekty deklarované vnútri procedúry (parametre, typy, premenné aj vnorené pro-

cedúry) existujú len počas behu procedúry. Každá procedúra vidí svoje lokálne premenné a navyše aj lokálne premenné všetkých procedúr, vnútri ktorých je deklarovaná (ak sa ich názvy líšia od názvov jej lokálnych premenných). Presne rovnako to funguje v Pascale.

Premenné sú pomenované identifikátormi, musia sa vytvoriť deklaráciou `var identifikator : typ;`. Pri vstupe do procedúry, v ktorej sú deklarované, majú nulovú hodnotu (v prípade poľa: všetky jeho prvky majú nulovú hodnotu) a predtým, ako premenná na konci procedúry zanikne, musí byť jej hodnota opäť nulová. Deklaráciu viac premenných toho istého typu môžeme skrátene zapísať `var i1, i2, ..., in : typ`.

Výrazy môžu obsahovať:

- konštanty (prir. čísla a konštanta `maxword` – najväčšie prir. číslo)
- premenné
- prvky polí (`pole[vyraz]`)
- aritmetické operácie, ktorých vstupom aj výstupom sú prirodzené čísla: $+$, $-$, $*$, **div** (celá časť podielu), **mod** (zvyšok po delení), **and**, **or**, **xor** (bitové operácie, definície viď vyššie) a **not** (prehodenie nulových a jednotkových bitov) – výsledky operácií sa automaticky berú modulo (`maxword`+1)
- relačné operácie (vstupom sú dve prirodzené čísla, výstupom bitová hodnota 1, ak relácia platí a 0, ak neplatí): $<$, $>$, $=$, $<=$, $>=$, $<>$
- zátvorky

Príkazy môžu byť nasledovných druhov:

- Blok: `begin` (príkazy oddelené bodkočiarkami) `end` – spôsobí postupné vykonanie príkazov, ktoré obsahuje, v danom poradí.
- Modifikačné príkazy: `premenna += vyraz` – spôsobí vyhodnotenie výrazu a jeho pripočítanie k premennej. Pritom `premenna` môže byť aj prvok poľa indexovaný nejakým výrazom. Premenná (resp. prvok poľa), ktorú príkaz modifikuje, sa už nikde v tomto príkaze nesmie vyskytnúť. Analogicky príkazy `-=` a `^=`.
- Vymieňací príkaz: `premenna =:= premenna` – vymení obsah dvoch premenných rovnakého typu. Ak sa jedná o prvky polí, nesmie sa žiadne z týchto polí používať vo výrazoch určujúcich indexy.
- Podmienený príkaz: `if podmienka then prikaz1 else prikaz2` – vyhodnotí sa podmienka (výraz s bitovým výsledkom), ak je výsledok 1, vykoná sa `prikaz1`, inak sa vykoná `prikaz2`. Platnosť podmienky sa vykonaním príslušného príkazu nesmie zmeniť. Časť `else` sa môže vypustiť, prípadné `else` sa vzťahuje k najbližšiemu neukončenému `if`.

- Príkaz cyklu: `for var identifikator = d to h do prikaz` – založí novú premennú daného mena, daný príkaz postupne vykonáva pre túto premennú nadobúdajúcu hodnoty `d`, `d`+1 až `h` a nakoniec riadiacu premennú opäť zruší. Hranice `d` a `h` sú celočíselné výrazy, ak `d`>`h`, cyklus sa ani raz nevykoná. Príkaz `prikaz` musí zachovávať hodnotu riadiacej premennej aj oboch hraníc cyklu (presnejšie: môže ich modi kovať, ale na konci prechodu cyklom musia mať tú istú hodnotu ako mali na začiatku príslušného prechodu). Takisto sa dá použiť konštrukcia `h downto d` namiesto `d to h`, potom bude riadiaca premenná postupne nadobúdať hodnoty `h`, `h`-1, až `d`.
- Volanie procedúry: `nazov_procedury ( par1, ..., parN )` – zavolá procedúru so zadanými parametrami. Parametre môžu byť premenné alebo prvky poľa (potom ale výraz, určujúci index prvku, musí mať po návrate z procedúry rovnakú hodnotu ako pred vstupom do nej). Počet parametrov aj ich typy musia zodpovedať deklarácii procedúry.
- Príkaz obrátenia výpočtu: `undo príkaz` – vykoná daný príkaz „odzadu“ podľa nasledujúcich pravidiel:

`undo begin`
`  p1; ...; pn`
`end`
	`begin`
	`  undo pn; ...; undo p1`
	`end`

`undo x += y`	`x -= y`
`undo x -= y`	`x += y`
`undo x ^= y`	`x ^= y`
`undo x =:= y`	`x =:= y`
`undo if x then y else z`	`if x then undo y else undo z`
`undo for x = d to h do p`	`for x = h downto d do undo p`
`undo P(x1; ...; xn)`	`undo` tela procedúry (`begin ... end`)
`undo undo p`	`p`

  Konštrukcia `begin p; undo p; end` teda nevykoná nič. (Aj keď počítať môže pomerne dlho.)
- Príkaz lokálneho výpočtu: `wrap prikaz1 on prikaz2` – skrátený zápis konštrukcie `begin prikaz1; prikaz2; undo prikaz1; end`.

Komentáre. Čokoľvek uzavreté v zložených zátvorkách (‘`{`’ a ‘`}`’) je komentár, ktorý počítač ignoruje (ako keby namiesto neho boli medzery). Komentár nesmie obsahovať zložené zátvorky.

Hlavný program nebudeme zavádzať. Aby sme sa vyhli problémom so vstupom a výstupom, budeme všetko programovať ako procedúry. Tie dostanú ako svoje parametre premenné, obsahujúce vstupné dáta a pre-

menné, ktoré majú byť predpísaným spôsobom modi kované podľa výstupu.

Časová a pamäťová zložitosť sú de nované podobne ako v klasickom programovaní: Časová zložitosť je počet vykonaných príkazov. Veľkosť pamäti využitej v danom okamihu spočítame ako súčet veľkostí všetkých lokálnych premenných (typy bit a word majú jednotkovú veľkosť, veľkosť poľa je súčet veľkostí jeho prvkov), počtu všetkých parametrov práve zavolaných procedúr a počtu práve zavolaných procedúr. Parametre sa bez ohľadu na ich typ počítajú ako jednotka, lebo sa odovzdávajú odkazom. Potom pamäťová zložitosť je maximum množstva využitej pamäti počas behu programu. Pozor! Keďže aj náš program je procedúra, jeho vstupy a výstupy sa do pamäťovej zložitosti započítavajú len ako konštanty, aj keď to môžu byť veľké polia.

Príklad

Procedúra na výpočet maxima zo zadaných `n` čísel. Dané je pole `X` celých čísel a premenná `max`, ku ktorej máme hľadané maximum pripočítať. To dokážeme takto: najskôr predpočítame pole `M`, kde `M[i]` bude maximum spomedzi čísel `X[1]` až `X[i]`. Potom pripočítame `M[n]` ku `max` a nakoniec `M[i]` opäť vyprázdnime. Časová aj pamäťová zložitosť sú lineárne, teda $O(n)$.

```
procedure Maximum(var n:word; var X:array [1..n] of word;
                  var max:word);
var M:array [0..n] of word;
begin
  wrap for var i=1 to n do
         if X[i]>M[i-1] then M[i] += X[i]
         else M[i] += M[i-1]
  on max += M[n];
end;
```

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

52. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Zadania 3. kola kategórie P

1. súťažný deň

Vydala IUVENTA
pre vnútornú potrebu Ministerstva školstva SR
Zodpovedný redaktor: M. Forišek
Sadzba programom LaTeX

MATEMATICKÁ OLYMPIÁDA
NA STREDNÝCH ŠKOLÁCH

52. ročník, školský rok 2002/2003

Zadania úloh 3. kola kategórie P

2. súťažný deň

P-III-4

Program: `poklad.pas/.c/.cpp`
Vstup: `poklad.in`
Výstup: `poklad.out`

Počas svojich pirátskych výprav nadobudol kapitán Flint značný majetok. Ako už piráti zvyknú robiť, rozhodol sa časť z neho zakopať na pustom ostrove. A tak sa aj stalo. Potom zobral ovčiu kožu v tvare konvexného N-uholníka a nakreslil na ňu cestu k pokladu. Potom mapu rozrezal na niekoľko kusov, pričom každý rez bola úsečka spájajúca dva vrcholy mnohouholníka a žiadne dva rezy sa vo vnútri mnohouholníka nepretínali.

Aby si poistil vernosť posádky svojho škuneru, rozhodol sa Flint darovať niektoré kusy mapy najšikovnejším pirátom. Aby sa však námorníci nemohli z mapy vyznať bez jeho pomoci, žiadni dvaja piráti nesmú dostať susedné diely mapy. Najviac koľko dielov mapy môže Flint rozdať pirátom?

Formát vstupu. Prvý riadok vstupného súboru obsahuje dve celé čísla N, M ($3 \leq N \leq 30\,000$, $0 \leq M \leq 30\,000$) oddelené medzerou. Pritom N je počet vrcholov mapy, M je počet rezov, ktoré Flint spravil. Nasleduje M riadkov, popisujúcich jednotlivé rezy. Každý z nich obsahuje dve celé čísla a_i, b_i oddelené medzerou – čísla vrcholov, cez ktoré viedol príslušný rez. Vrcholy sú očíslované od 1 do N po obvode mnohouholníka.

Formát výstupu. Výstupný súbor má obsahovať jeden riadok a na ňom jedno celé číslo – najväčší počet kusov mapy, ktoré sa dajú rozdať pirátom (za podmienky, že žiadni dvaja nesmú dostať susedné kusy).

Príklad:

Vstup	**Výstup**

`11 4`
`1 3`
`5 7`
`10 8`
`4 11`
	`3`

P-III-5

Program: `vahy.pas/.c/.cpp`
Vstup: `vahy.in`
Výstup: `vahy.out`
Knižnica: `vahy_lib`

V nemenovanom kráľovstve nedávno vyhlásili konkurz na kráľovského radcu. Podmienky sú veľmi náročné: Každý uchádzač dostane N mincí, ktoré môžu a nemusia mať navzájom rôzne hmotnosti. Jeho úlohou bude roztriediť tieto mince na niekoľko kôpok, pričom mince na jednej kôpke musia mať navzájom rovnakú hmotnosť a následne usporiadať kôpky podľa hmotností mincí tak, aby na prvej kôpke boli najľahšie mince, na druhej ťažšie, atď., až na poslednej najťažšie mince. K dispozícií bude mať iba dvojramenné váhy, na ktorých môže porovnať hmotnosť ľubovoľných dvoch mincí. (Pri každom vážení musí dať na každú misku práve jednu mincu.)

Vašou úlohou je napísať program, ktorý by tento konkurz vyhral. Očíslujeme si mince číslami od 1 do N. Váš program bude môcť volať funkciu `porovnaj`, ktorá porovná váhy dvoch mincí a programu oznámi výsledok. Váš program musí samozrejme nájsť odpoveď čo najrýchlejšie. Pritom musí spraviť všetky *nutné* váženia (nesmú existovať dve rôzne riešenia konzistentné s váženiami, ktoré spravil váš program). Navyše váš program nesmie spraviť žiadne *zbytočné* váženie, t.j. také, ktorého výsledok by vyplýval zo skôr uskutočnených vážení.

Popis funkcie `porovnaj`

Funkcia `porovnaj` je de novaná v knižnici `vahy_lib`. Aby ju váš program mohol volať, musí obsahovať nasledujúci riadok:

Pascal: `uses vahy_lib;`
C/C++: `#include "vahy_lib.h"`

Hlavička funkcie `porovnaj` vyzerá nasledovne:

Pascal: `function porovnaj(a,b: longint): integer;`
C/C++: `int porovnaj(int, int);`

Táto funkcia očakáva ako vstupné parametre čísla dvoch mincí. Vráti hodnotu -1, ak je prvá minca ľahšia, $+1$, ak je ťažšia a 0, ak sú obe rovnako ťažké.

Nezabudnite, že váš program **nesmie** volať funkciu `porovnaj` zbytočne, t.j. výsledok žiadneho volania nesmie byť jednoznačne určený predchádzajúcimi volaniami. Napr. ak sme už zistili, že minca 1 je ľahšia ako

minca 2 a že mince 2, 3 majú rovnakú hmotnosť, nesmieme už zavolať `porovnaj` s parametrami 1 a 3. Okrem toho, váš program môže funkciu `porovnaj` zavolať najviac 250 000-krát.

Formát vstupu. Vstupný súbor obsahuje jediný riadok a na ňom jedno celé číslo N ($1 \leq N \leq 10\,000$) – počet mincí.

Formát výstupu. Výstupný súbor má obsahovať K riadkov, kde K je počet rôznych hmotností mincí. Na každom riadku budú uvedené čísla mincí z príslušnej kôpky. Teda na prvom riadku budú čísla najľahších mincí, atď., až na poslednom najťažších. V každom riadku musia byť tieto čísla uvedené v rastúcom poradí a oddelené práve jednou medzerou.

Príklad:

Vstup	**Priebeh komunikácie**	**Výstup**
`4`	Volanie `porovnaj(2,4)` vrátilo -1.	`2`
	Volanie `porovnaj(1,2)` vrátilo $+1$.	`1 3`
	Volanie `porovnaj(3,4)` vrátilo -1.	`4`
	Volanie `porovnaj(1,3)` vrátilo 0.	

SLOVENSKÁ KOMISIA MATEMATICKEJ OLYMPIÁDY

52. ROČNÍK MATEMATICKEJ OLYMPIÁDY

Zadania 3. kola kategórie P

2. súťažný deň

Vydala IUVENTA
pre vnútornú potrebu Ministerstva školstva SR
Zodpovedný redaktor: M. Forišek
Sadzba programom LaTeX