IX MiĆędzynarodowa Olimpiada Informatyczna Kapsztad, grudzieĆń 1997

teksty zadaĆń

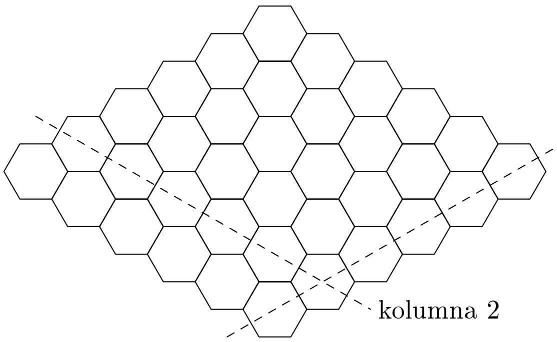
Krzysztof Diks, Andrzej Walat
przekład

Gra Hex

Celem gry dla pierwszego gracza jest ustawienie swoich pionów na planszy w taki sposób, aby „związać” kolumnę 1 z kolumną N.

Reguły gry

Hex *jest strategiczną grą dwuosobową rozgrywaną na planszy w kształcie rombu o rozmiarach $N \times N$. Pola na planszy są sześciokątami foremnymi. Rysunek przedstawia planszę dla $N = 6$.*



(1) Graczami są Twój program i biblioteka testująca.

(2) Twój program zawsze wykonuje pierwszy ruch.

(3) Gracze na zmianę umieszczają swoje piony na planszy.

(4) Piona można postawić tylko na wolnym polu.

(5) Dwa pola są sąsiednie, jeśli mają wspólną krawędź.

(6) Piony tego samego gracza na sąsiednich polach sa „związane”.

(7) Relacja wiązania jest przechodnia (i przemienna): jeśli pion1 jest związany z pionem2 i pion2 jest związany z pionem3, to pion 1 jest związany z pionem3.

Zadanie

- *Napisz program grający w Hex.*
- *Celem gracza 1 (Twojego programu) jest związanie swojego piona w kolumnie 1 ze swoim pionem w kolumnie N.*
- *Drugi gracz (program testujący) próbuje związać swój pion z wiersza 1 ze swoim pionem z wiersza N.*
- *Jeśli Twój program gra optymalnie, to zawsze wygra.*

Wejście i wyjście

Twój program nie może ani czytać z żadnego pliku, ani pisac do żadnego pliku. Nie może czytać z klawiatury, ani wypisywać niczego na ekran. Wszystkie dane wejściowe będzie otrzymywał za pomocą funkcji z biblioteki HexLib. Ta biblioteka utworzy plik HEX.OUT, którego zawartość powinieneś zignorować.

Na początku Twojej rozgrywki z komputerem stan planszy przedstawia stan gry (pewne pola planszy mogą być zajęte), którą Twój program może jeszcze wygrać. Twój program musi korzystać z funkcji `GetMax` *i* `LookAtBoard`*, by ustalić stan gry. Na początku rozgrywki liczby pionów obu graczy na planszy są takie same.*

Ograniczenia

(1) *Rozmiar planszy (N) będzie zawsze należał do przedziału od 1 do 20 (włącznie).*

(2) *Twój program może potrzebować do 200 ruchów, by zakończyć grę. Cała rozgrywka musi się zakończyć w 40 sekund. Gwarantuje się, że program testujący wykorzysta co najwyżej 20 sekund.*

Biblioteka

Gotową bibliotekę HexLib musisz skonsolidować ze swoim kodem. Dla każdego języka programowania, w katalogu zadania znajduje się przykładowy plik (`TESTHEX.CPP`*,* `TESTHEX.C`*,* `TESTHEX.PAS` *i* `TESTHEX.BAS`*) pokazujący, jak to zrobić.*

Biblioteka HexLib zawiera następujące funkcje i procedury (odpowiednio: Pascal, C/C++ i Basic):

```
function LookAtBoard (row, column: integer): integer;
int LookAtBoard (int row, int column);
declare function LookAtBoard cdecl (byval x as integer, byval y
as integer)
```

Zwraca

- -1, *jeśli* $row < 1$, *lub* $row > N$, *lub* $column < 1$, *lub* $column > N$,
- *0, jeśli nie ma piona na tej pozycji,*
- *1, jeśli pion na danej pozycji należy do Twojego programu (gracz 1),*
- *2, jeśli pion na danej pozycji należy do programu testującego (gracz 2).*

```
procedure PutHex (row, column: integer);
void PutHex (int row, int column);
declare sub PutHex cdecl (byval x as integer, byval y as integer)
```

Umieszcza Twojego piona na wskazanej pozycji, jeśli jest wolna.

```
function GameIsOver: integer;
int GameIsOver (void);
declare function GameIsOver cdecl ()
```

Zwraca jedną z następujących wartości

- *0 — gra się nie zakończyła,*
- *1 — na każdym polu planszy stoi pion,*
- *2 — Twój program wygrał,*
- *3 — program testujący wygrał.*

```
procedure MakeLibMove;
void MakeLibMove(void);
declare sub MakeLibMove cdecl ()
```

Umożliwia programowi testującemu obliczenie następnego ruchu i umieszczenie swojego piona na planszy. Zmiana na planszy będzie pokazywana przez `LookAtBoard` *i inne funkcje.*

```
function GetRow: integer;
int GetRow (void);
declare function GetRow cdecl ()
```

Zwraca numer wiersza, w którym program testujący umieścił piona, lub -1*, jeśli żaden pion nie został jeszcze postawiony. Ta funkcja zawsze zwraca tę samą wartość, aż do ponownego wywołania* `MakeLibMove`*.*

```
function GetColumn: integer;
int GetColumn (void);
declare function GetColumn cdecl ()
```

Zwraca numer kolumny, w której program testujący umieścił piona, lub -1*, jeśli żaden pion nie został jeszcze postawiony. Ta funkcja zawsze zwraca tę samą wartość, aż do ponownego wywołania* `MakeLibMove`*.*

```
function GetMax: integer;
int GetMax (void);
declare function GetMax cdecl ()
```

Zwraca rozmiar planszy N.

Punktacja

- *Jeśli Twój program wygra, zdobywasz całą pulę punktów za rozgrywkę.*
- *Jeśli Twój program przegra, zdobywasz 20% puli punktów za rozgrywkę.*
- *Jeśli Twój program zatrzyma się przed zakończeniem gry lub przekroczy limit czasu, otrzymujesz 0 punktów.*

Krzysztof Diks, Andrzej Walat
przekład

Rozpoznawanie znakow

Ten problem wymaga napisania programu rozpoznającego znaki.

Szczegóły

Każdy idealny obraz znaku składa sie z 20-tu wierszy, po 20 cyfr '0' lub '1' w każdym wierszu. Spójrz na układ obrazów znaków w pliku na rys. 1a.

Plik FONT.DAT zawiera reprezentacje 27 idealnych obrazów znaków w następującym porządku:

␣abcdefghijklmnopqrstuvwxyz

gdzie ␣ reprezentuje znak odstępu.

Plik IMAGE.DAT zawiera jeden lub więcej, być może zakłóconych, obrazów znaków. Obraz znaku może być zakłócony na następujące sposoby:

- *co najwyżej jeden wiersz może być zdublowany (zdublowane wiersze występują jeden po drugim),*
- *co najwyżej jeden wiersz może zostać opuszczony,*
- *pewne zera mogły zostać zamienione na jedynki,*
- *pewne jedynki mogły zostać zamienione na zera.*

W żadnym obrazie znaku nie występuje jednocześnie zdublowanie i opuszczenie wiersza. W każdym obrazie znaku w testach liczba przekłamanych zer i jedynek nie przekracza 30%.

W przypadku zdublowania wiersza, w każdym z obu wierszy mogą wystąpić przekłamania znaków, niekoniecznie w tym samym miejscu.

Zadanie

Napisz program rozpoznający (na podstawie wzorców czcionek zapisanych w pliku FONT.DAT) ciąg złożony z jednego lub więcej znaków w obrazie zapisanym w pliku IMAGE.DAT.

Rozpoznając znaki w obrazie dobieraj je w taki sposób, żeby było możliwe przekształcenie ciągu odpowiadających im wzorców czcionek na dany (zakłócony) obraz za pomocą minimalnej łącznej liczby przestawień cyfr '1' lub '0'. Możesz przy tym przyjąć najbardziej korzystne założenie dotyczące dublowania i opuszczania wierszy. W przypadku zdublowania wierszy licz zakłócenia tylko w mniej zniekształconym wierszu. Dobrze napisany program powinien rozpoznać wszystkie znaki w danych testowych. Istnieje jedno najlepsze rozwiązanie dla każdego zestawu danych testowych.

Poprawne rozwiązanie wykorzysta dokładnie wszystkie dane zapisane w pliku IMAGE.DAT.

Wejście

Oba pliki danych zaczynają się od liczby całkowitej N ($19 \leqslant N \leqslant 1200$), która określa liczbę następnych wierszy pliku:

```
N
(znak-1)(znak-2)(znak-3)...(znak20)
(znak-1)(znak-2)(znak-3)...(znak20)
(znak-1)(znak-2)(znak-3)...(znak20)
(znak-1)(znak-2)(znak-3)...(znak20)
(znak-1)(znak-2)(znak-3)...(znak20)
(znak-1)(znak-2)(znak-3)...(znak20)
(znak-1)(znak-2)(znak-3)...(znak20)
```

```
(znak-1)(znak-2)(znak-3)...(znak20)

(znak-1)(znak-2)(znak-3)...(znak20)

(znak-1)(znak-2)(znak-3)...(znak20)

...
```

Każdy z tych wierszy zawiera 20 znaków. Nie ma żadnych odstępów oddzielających zera i jedynki.

Plik FONT.DAT zawiera wzorce czcionek i zawsze ma 541 wierszy. W każdym teście jego zawartość może być inna.

Punktacja

Punkty będą procentem rozpoznanych znaków.

Wyjście

Twój program musi utworzyć plik IMAGE.OUT, zawierający jeden napis złożony z rozpoznanych znaków. Powinien on mieć format jednego wiersza tekstu ASCII. W pliku wyjściowym nie powinno być żadnych separatorów. Jeśli Twój program nie może rozpoznać jakiegoś znaku, powinien na odpowiedniej pozycji wypisać „?”.

Przykładowe pliki danych: przedstawiono niekompletny przykład pokazujący początek pliku FONT.DAT (spacja i `a`*) oraz przykład pliku IMAGE.DAT (zakłócone* `a`*).*

FONT.DAT

```
540
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000011100000000000
00000111111011000000
00001111111001100000
00001110001100100000
00001100001100010000
00001100000100010000
00000100000100010000
00000010000000110000
00000001000001110000
00001111111111110000
00001111111111110000
00001111111111000000
00001000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Rys. 1a

IMAGE.DAT

```
19
00000000000000000000
00000000000000000000
00000000000000000000
00000011100000000000
00100111011011000000
00001111111001100000
00001110001100100000
00001100001100010000
00001100000100010000
00000100000100010000
00000010000000110000
00001111011111110000
00001111111111110000
00001111111111000000
00001000010000000000
00000000000000000000
00000000000001000000
00000000000000000000
00000000000000000000
```

Rys. 1b

Przykładowe wyjście:

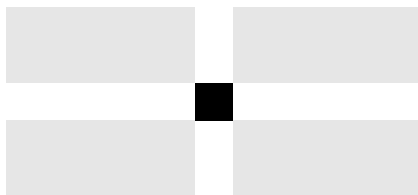
IMAGE.OUT	**Objaśnienie**
a	*Rozpoznano pojedynczy znak* `a`

Krzysztof Diks, Andrzej Walat
przekład

Opisywanie mapy

Jesteś pomocnikiem kartografa i otrzymałeś zadanie naniesienia nazw miast na nowej mapie.

Mapa jest siatką 1000 × 1000 pól. Każde miasto zajmuje jedno pole na mapie. Nazwy miast należy umieścić na mapie w prostokątnych ramkach utworzonych z pól siatki. Taką ramkę nazywamy etykietą.



Rys. 1 Miasto i cztery dozwolone położenia etykiety

Rozmieszczenie etykiet musi spełniać następujące warunki:

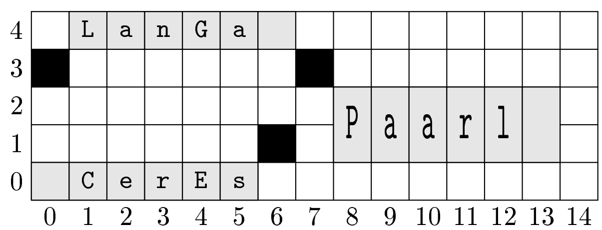
(1) *Etykieta miasta musi zajmować jedną z czterech pozycji wskazanych na rysunku 1.*

(2) *Etykiety nie mogą na siebie nachodzić.*

(3) *Etykiety nie mogą nachodzić na pola reprezentujące miasta.*

(4) *Etykiety nie mogą wychodzić poza mapę.*

Każda etykieta zawiera wszystkie litery nazwy miasta plus pojedynczą spację. Dla każdej nazwy miasta dana jest szerokość i wysokość tworzacych ją znaków; pojedyncza spacja ma taki sam rozmiar.



Rys. 2 Wycinek mapy

Kolumny siatki są ponumerowane od lewej do prawej kolejnymi liczbami, poczynając od 0. Wiersze są ponumerowane od dołu do góry, również poczynając od 0.

Rysunek 2 przedstawia lewy dolny róg mapy z miastami Langa na pozycji (0,3), Ceres na pozycji (6,1) i Pearl na pozycji (7,3). Wszystkie etykiety są umieszczone prawidłowo, ale nie jest to jedyne prawidłowe ich rozmieszczenie.

Zadanie

Napisz program, który wczytuje pozycje miast na mapie wraz z wielkościami liter i nazwami tych miast. Twój program powinien rozmieścić nazwy na mapie zgodnie z podanymi warunkami i wypisać pozycje etykiet.

Wejście

Pierwszy wiersz pliku wejściowego zawiera liczbę miast na mapie (N). W każdym z kolejnych N wierszy są zapisane następujące dane: pozycja miasta na mapie – numer kolumny (X) i numer wiersza (Y); rozmiar liter tworzących nazwę miasta — szerokość (W) i wysokość (H), które są liczbami całkowitymi; nazwa miasta. Nazwy miast są pojedynczymi słowami. Liczba miast jest nie większa niż 1000. Nazwa miasta nie może mieć więcej niż 200 znaków.

Przykładowe wejście:

MAPS.DAT	**Objaśnienie:**
3	$N = 3$
0 3 1 1 Langa	$X = 0, Y = 3, W = 1, H = 1$
6 1 1 1 Ceres	
7 3 1 2 Paarl	

Wyjście

Twój program powinien zapisać N wierszy w pliku wyjściowym. Każdy wiersz odpowiada jednemu miastu z pliku wejściowego i musi zawierać: numer kolumny i numer wiersza górnego lewego pola etykiety tego miasta. Jeśli Twój program nie może ulokować etykiety, to musi wypisać parę liczb -1 -1*. Porządek wierszy w pliku wyjściowym odpowiada porządkowi miast w pliku wejściowym.*

Przykładowe wyjście:

MAPS.OUT	**Objaśnienie:**
1 4	*etykieta miasta Langa na pozycji* $(1, 4)$
0 0	*etykieta miasta Ceres na pozycji* $(0, 0)$
8 2	*etykieta miasta Paarl na pozycji* $(8, 2)$

Punktacja

Dla każdego zestawu danych testowych:

- *Wynik punktowy będzie procentem liczby prawidłowo rozmieszczonych etykiet w stosunku do wzorcowego rozwiązania organizatorów.*
- *Minimalna liczba punktów to 0%, maksymalna 100%.*
- *Jeśli rozmieszczenie jakiejkolwiek etykiety narusza ograniczenia, Twój program otrzyma 0 pkt.*
- *Jeśli etykiety nie odpowiadają miastom, Twój program otrzyma 0 pkt.*

Krzysztof Diks, Andrzej Walat
przekład

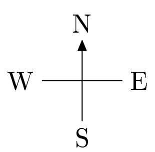
Wyprawa na Marsa

W ramach przygotowań do misji kosmicznej planuje się, że na powierzchni Marsa wyląduje pojemnik zawierający pewną liczbę pojazdów marsjańskich.

Wszystkie te pojazdy wyruszą z miejsca lądowania pojemnika i będą się kierowały do nadajnika, który wyląduje w pobliżu. Pojazdy przemieszczając się w kierunku nadajnika mają za zadanie pobierać próbki skał marsjańskich. Każda próbka może być pobrana tylko raz, przez pierwszy pojazd, który ją napotka. Inne pojazdy mogą przejeżdżać przez to samo miejsce.

Pojazdy nie mogą wjeżdżać na niebezpieczne obszary.

Pojazd marsjański jest skonstruowany w taki sposób, że może poruszać się tylko w dwóch kierunkach — południowym lub wschodnim, a jego trasa musi przebiegać przez pola prostokątnej siatki, od miejsca wylądowania pojemnika do miejsca, w którym znajduje się nadajnik. Na tym samym polu może się w tej samej chwili znajdować wiele pojazdów.



pojemnik →	0	0	0	0	0	0	0	0	0	0	wiersz 1
	0	0	0	0	0	1	1	0	0	0	
	0	0	0	1	0	2	0	0	0	0	
	1	1	0	1	2	0	0	0	0	1	
	0	1	0	0	2	0	1	1	0	0	
	0	1	0	1	0	0	1	1	0	0	
	0	1	2	0	0	0	0	1	0	0	
	0	0	0	0	0	0	0	0	0	0	wiersz 8

nadajnik

UWAGA: Jeśli pojazd nie może wykonać dozwolonego ruchu zanim osiągnie nadajnik, wszystkie pobrane przez niego próbki są stracone bezpowrotnie.

Zadanie

Zaplanuj ruchy pojazdów tak, aby uzyskać jak najwięcej punktów, maksymalizując liczbę zebranych próbek skalnych oraz liczbę pojazdów marsjańskich, które dotrą do nadajnika.

Wejście

Powierzchnię planety pomiędzy pojemnikiem i nadajnikiem reprezentuje siatka o rozmiarach $P \times Q$, na której pojemnik zajmuje zawsze pole $(1, 1)$, a nadajnik — pole (P, Q). Rodzaj pola określa się następująco:

- *Obszar czysty: 0*
- *Obszar niebezpieczny: 1*
- *Próbka skały: 2*

Plik wejściowy zawiera następujące dane:

```
liczba_pojazdów_w_pojemniku
P
Q
(X1 Y1) (X2 Y1) (X3 Y1)... (XP Y1)
(X1 Y2) (X2 Y2) (X3 Y2)... (XP Y2)
(X1 Y3) (X2 Y3) (X3 Y3)... (XP Y3)
...
(X1 YQ) (X2 YQ) (X3 YQ)... (XP YQ)
```

P i Q to rozmiary siatki (liczba kolumn i liczba wierszy) nie większe niż 128, liczba_pojazdów_w_pojemniku jest liczbą całkowitą mniejszą od 1000, każdy z Q wierszy reprezentuje jeden wiersz siatki. Przykładowe wejście:

MARS.DAT	**Objaśnienie:**
2	*Liczba pojazdów*
10	*P*
8	*Q*
0 0 0 0 0 0 0 0 0 0	*Wiersz 1*
0 0 0 0 0 1 1 0 0 0	*Wiersz 2*
0 0 0 1 0 2 0 0 0 0	*Wiersz 3*
1 1 0 1 2 0 0 0 0 1	*Wiersz 4*
0 1 0 0 2 0 1 1 0 0	*Wiersz 5*
0 1 0 1 0 0 1 1 0 0	*Wiersz 6*
0 1 2 0 0 0 0 1 0 0	*Wiersz 7*
0 0 0 0 0 0 0 0 0 0	*Wiersz 8*

Wyjście

Kolejne wiersze opisują ruchy pojazdów w kierunku nadajnika. Każdy wiersz składa się z numeru pojazdu i cyfry 0 lub 1, gdzie 0 oznacza ruch na południe, a 1 na wschód. Przykładowe wyjście:

MARS.OUT	**Objaśnienie:**
1 1	*pojazd 1 rusza na wschód*
1 0	*pojazd 1 rusza na południe*
2 1	*pojazd 2 rusza na wschód*
2 0	*pojazd 2 rusza na południe*
1 1	*...*
1 1	
2 0	
2 1	
2 0	
2 0	
2 0	
2 0	
1 1	
1 0	
1 0	
1 0	
1 0	
1 0	
1 0	
2 0	
2 1	
1 1	
1 1	
1 1	
1 1	
1 1	
2 1	
2 1	
2 1	
2 1	
2 1	
2 1	

Do nadajnika dotarły oba pojazdy dostarczając 3 próbki skał. Za takie rozwiązanie otrzymałbyś maksymalną liczbę punktów (100%).

Punktacja

Punkty zostaną obliczone na podstawie liczby próbek zebranych i dostarczonych do nadajnika z uwzględnieniem liczby pojazdów, które dotarły do nadajnika i liczby tych, które nie dotarły.

- *Każdy nielegalny ruch unieważnia rozwiązanie testu i daje 0 punktów za test. Nielegalny ruch występuje wtedy, gdy pojazd wjeżdża na niebezpieczne pole lub opuszcza siatkę.*
- *Wynik = (liczba próbek dostarczonych do nadajnika + liczba pojazdów, które dotarły do nadajnika − liczba pojazdów, które nie dotarły do nadajnika) procentowo w stosunku do maksymalnej liczby punktów.*
- *Maksymalnie można zdobyć 100%, a minimalnie 0%.*

Krzysztof Diks, Andrzej Walat
przekład

Baza kontenerowa

Kompania Neptune Cargo zarządza magazynem kontenerów. Magazyn przyjmuje kontenery na przechowanie, a następnie je wydaje.

Kontenery przybywają do magazynu o równych godzinach. Pozostają w magazynie całkowitą liczbę godzin. Dokumenty przybywające z kontenerem określają deklarowany czas jego wydania. Pierwszy kontener przybywa o godz. 1. Zlecenie wydania kontenera może nastąpić w dowolnym czasie różniącym się od deklarowanego nie więcej niż 5 godz.

W tym zadaniu czas jest odmierzany w godzinach — w rosnących, całkowitych liczbach dodatnich, nie większych niż 150.

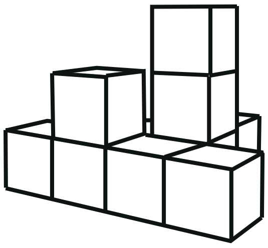
Dźwig, który może swobodnie poruszać się nad całą powierzchnią magazynu, umieszcza kontenery w magazynie, usuwa je z magazynu, a czasami przemieszcza.

Zadanie

Napisz program oparty na dobrej strategii przyjmowania, przemieszczania i wydawania kontenerów. Dobra strategia minimalizuje liczbę ruchów dźwigu.

Magazyn ma kształt prostopadłościanu. Jego długość (X), szerokość (Y) i wysokość (Z) są dostępne dla programu.

Każdy kontener jest sześcianem o rozmiarach $1 \times 1 \times 1$. Kontenery mogą być umieszczane na podłodze lub w stertach jeden na drugim. Dźwig może przemieścić tylko kontener znajdujący się na wierzchu.



Przemieszczenie kontenera z jednego miejsca na inne jest traktowane jako jeden ruch dźwigu. Wszystkie ruchy dźwigu mają miejsce pomiędzy pobraniami i wydaniami kontenerów. Czas ruchu jest zaniedbywalny.

Gdy magazyn jest pełny, Twój program musi odmówić przyjęcia nowego kontenera. Twój program może się stać mniej efektywny lub być niezdolny do działania, gdy magazyn jest prawie pełny. Twój program ma prawo w dowolnej chwili odmówić przyjęcia kontenera.

Wejście

Twój program powinien współpracować interakcyjnie z modułem symulacyjnym, dostarczającym dane, na które Twój program musi reagować, podejmując odpowiednie akcje i wysyłając komunikaty. W momencie rozpoczęcia działania programu magazyn jest pusty.

Gdy będziesz testował program, moduł dostarczy sensownych danych dla ustalonego zestawu danych o małym rozmiarze. Każdy kontener jest jednoznacznie identyfikowany za pomocą dodatniej liczby całkowitej. Twój program może w każdej chwili wywołać następujące funkcje i procedury:

```
int GetX();
function GetX: integer;
DECLARE FUNCTION GetX CDECL ()
```

Zwraca długość magazynu (integer).

```
int GetY();
function GetY: integer;
DECLARE FUNCTION GetY CDECL ()
```

Zwraca szerokość magazynu (integer).

```
int GetZ();
function GetZ: integer;
DECLARE FUNCTION GetZ CDECL ()
```

Zwraca wysokość magazynu (integer).

X, Y, Z nie przekraczają 32.

Następujące funkcje dostarczają informacji o sekwencji zdarzeń (przybywanie i wydawanie kontenerów). Kontenery przybywają o równych godzinach, zlecenia wydania są przyjmowane w trakcie godzin. Tak więc dla celów odmierzania czasu każde przybycie kontenera oznacza minięcie jednej godziny.

```
int GetNextContainer();
function GetNextContainer: integer;
DECLARE FUNCTION GetNextContainer CDECL ()
```

Zwraca numer kolejnego kontenera, który ma być przyjęty lub wydany. Jeśli nie ma więcej kontenerów do przyjęcia lub wydania, zwraca 0, co oznacza, że Twój program powinien zakończyć działanie, nawet gdy magazyn jest niepusty.

```
int GetNextAction();
function GetNextAction: integer;
DECLARE FUNCTION GetNextAction CDECL ()
```

Zwraca liczbę całkowitą wyznaczającą akcję, którą trzeba podjąć: 1 oznacza przyjęcie kontenera, 2 — wydanie kontenera.

```
int GetNextStorageTime();
function GetNextStorageTime: integer;
DECLARE FUNCTION GetNextStorageTime CDECL ()
```

Zwraca deklarowany czas wydania kontenera w godzinach (liczony od początku). Ta wartość ma służyć programowi dla celów planowania; rzeczywiste zlecenie wydania kontenera może nastąpić w dowolnym czasie różniącym się od deklarowanego co najwyżej o 5 godzin. Ta funkcja daje sensowną wartość tylko wtedy, gdy `GetNextAction` *zwraca 1.*

Kolejność wywoływania tych trzech funkcji jest dowolna. Kolejne wywołania `GetNextContainer`, `GetNextAction` *i* `GetNextStorageTime` *dają zawsze informację o tym samym kontenerze, aż do momentu przyjęcia, wydania lub odmowy przyjęcia kontenera; po czym, te funkcje dają informację o następnym kontenerze.*

Wyjście

Gdy Twój program uzyska wszystkie potrzebne informacje o kolejnym kontenerze, powinien skorzystać z nastepujacyh funkcji symulujących operacje magazynowe.

```
int MoveContainer(int x1, int y1, int x2, int y2);
function MoveContainer(x1, y1, x2, y2: integer): integer;
DECLARE FUNCTION MoveContainer CDECL (BYVAL x1 AS INTEGER, BYVAL
y1 AS INTEGER, BYVAL x2 AS INTEGER, BYVAL y2 AS INTEGER)
```

Przenieś kontener z wierzchu sterty $x1, y1$ na wierzch sterty $x2, y2$. Zwraca 1, jeśli ta akcja jest dozwolona, 0 gdy jest niedopuszczalna.

```
void RefuseContainer();
procedure RefuseContainer;
DECLARE SUB RefuseContainer CDECL ()
```

Odmowa przyjęcia kontenera.

```
void StoreArrivingContainer(int x, int y);
procedure StoreArrivingContainer(x, y: integer);
DECLARE SUB StoreArrivingContainer CDECL (BYVAL x AS INTEGER,
BYVAL y AS INTEGER)
```

Przyjmuje kontener i umieszcza na wierzchu sterty w pozycji x, y.

```
void RemoveContainer(int x, int y);
procedure RemoveContainer(x, y: integer);
DECLARE SUB RemoveContainer CDECL (BYVAL x AS INTEGER, BYVAL y AS
INTEGER)
```

Wydaje z magazynu kontener znajdujący się na wierzchu sterty x, y.

Jeśli Twój program nie może wykonać zleconej akcji, powinien zakończyc działanie. Niedozwolone ruchy są ignorowane przez moduł i nie mają wpływu na symulację, ani na punktację.

Twój program nie powinien tworzyć żadnego pliku. Moduł współdziałający z Twoim programem tworzy plik akcji, który będzie podstawą oceny.

Kolejność akcji

Twój program powinien otrzymać informację o zleceniu dotyczącym kontenera. Następnie, jeśli trzeba, powinien przemieścić kontenery w magazynie i przyjąć kontener, wydać kontener lub odmówić wykonania zlecenia.

Biblioteka

Gotową bibliotekę StackLib musisz skonsolidować ze swoim kodem. Standardowe biblioteki C i C++ zawierają tę bibliotekę i będzie ona automatycznie konsolidowana z Twoim programem, jeśli włączysz odpowiedni plik nagłówkowy. Przykładowe pliki źródłowe znajdują się w katalogu zadania. Są to `TESTSTK.BAS`, `TESTSTK.PAS`, `TESTSTK.CPP` *i* `TESTSTK.C`.

Punktacja

Program będzie testowany na kilku zestawach danych i w każdym przypadku punkty będą przyznawane na podstawie porównania z najlepszym znanym nam programem, z uwzględnieniem następujacych wskaźników:

- *Łączna liczba ruchów dźwigu.*
- *Kara 5 ruchów za każdą odmowę przyjęcia kontenera.*
- *Kara 5 ruchów za każdy kontener, który nie był przechowywany w magazynie (bo np. program zakończył w normalny sposób swoje działanie przed wykonaniem całego zadania).*
- *Suma punktów będzie obliczona względem najlepszego znanego rozwiązania.*
- *Jeśli program wykona dwa razy więcej operacji niż potrzeba, otrzyma 0 pkt.*
- *Minimalny wynik to 0%, a maksymalny to 100%.*

Krzysztof Diks, Andrzej Walat
przekład

Żarłoczne iShongololo

iShongololo *to zuluska nazwa krocionoga. Jest to długi, błyszczący, czarny owad, mający wiele nóg.*

iShongololo żywi się jadalnymi „owocami”. Na potrzeby tego zadania przyjmujemy, że mają one kształt prostopadłościanu o całkowitoliczbowych rozmiarach L (length — długość), W (width — szerokość) i H (height — wysokość).

Zadanie

Napisz program, który maksymalizuje liczbę bloków zjadanych przez iShongololo bez naruszenia ustalonych ograniczeń. Program musi wypisywać czynności wykonywane przez iShongololo, gdy drąży swoją ścieżkę zjadając owoc.

Na początku iShongololo znajduje się na zewnątrz owocu. Pierwszym blokiem, który musi zjeść jest 1, 1, 1. Następnie musi wejść do tego bloku. iShongololo kończy, gdy nie może już zgodnie z regułami zjeść żadnego bloku, ani wykonać ruchu.

Ograniczenia

(1) iShongololo zajmuje dokładnie jeden pusty blok.

(2) iShongololo zjada na raz jeden pełny blok.

(3) iShongololo nie może wykonać ruchu na pozycję, na której już był (tzn. zawrócić lub przeciąć swoją ścieżkę).

(4) iShongololo nie może wejść do pełnego (nie zjedzonego) bloku lub wyjść na zewnątrz owocu.

(5) iShongololo może wykonać ruch tylko do bloku o przyległej ścianie; może też zjeść blok o przyłeglej ścianie, ale tylko taki, który nie ma wspólnej ściany z żadnym innym pustym blokiem.

Wejście

Na wejściu Twój program otrzyma trzy liczby całkowite — długość (L), szerokość (W) i wysokość (H) prostopadłościanu.

Każda z liczb L, W, H jest podana w osobnym wierszu i należy do przedziału od 1 do 32 (włącznie). Przykładowe wejście:

TOXIC.DAT	**Objaśnienie:**
2	*długość — 2*
3	*szerokość — 2*
2	*wysokość — 2*

Wyjście

Plik wyjściowy składa się z wierszy rozpoczynających się od litery `E` *(eat — zjada) lub* `M` *(move — wchodzi do), po której występują 3 liczby całkowite reprezentujące współrzędne bloku (odpowiednio L, W, H), który iShongololo zjada lub do którego wchodzi.*

Oto przykład poprawnego pliku wyjściowego (być może nie reprezentującego rozwiązania optymalnego) dla przykładowych danych.

TOXIC.OUT	**Objaśnienie:**
E 1 1 1	*Zjada blok 1 1 1*
M 1 1 1	*Wchodzi do bloku 1 1 1*
E 2 1 1	*Zjada blok 2 1 1*
E 1 1 2	*Zjada blok 1 1 2*
E 1 2 1	*Zjada blok 1 2 1*
M 1 2 1	*Wchodzi do bloku 1 2 1*
E 1 3 1	*Zjada blok 1 3 1*
M 1 3 1	*Wchodzi do bloku 1 3 1*
E 2 3 1	*Zjada blok 2 3 1*
E 1 3 2	*Zjada blok 1 3 2*
M 1 3 2	*Wchodzi do bloku 1 3 2*

Punktacja

- *Jeśli iShongololo naruszy ograniczenia, otrzymasz 0 punktów za test.*
- *Ogólna liczba punktów wyraża procentowo stosunek liczby zjedzonych bloków do najlepszego znanego rozwiązania.*
- *Nie możesz otrzymać więcej niż 100%.*