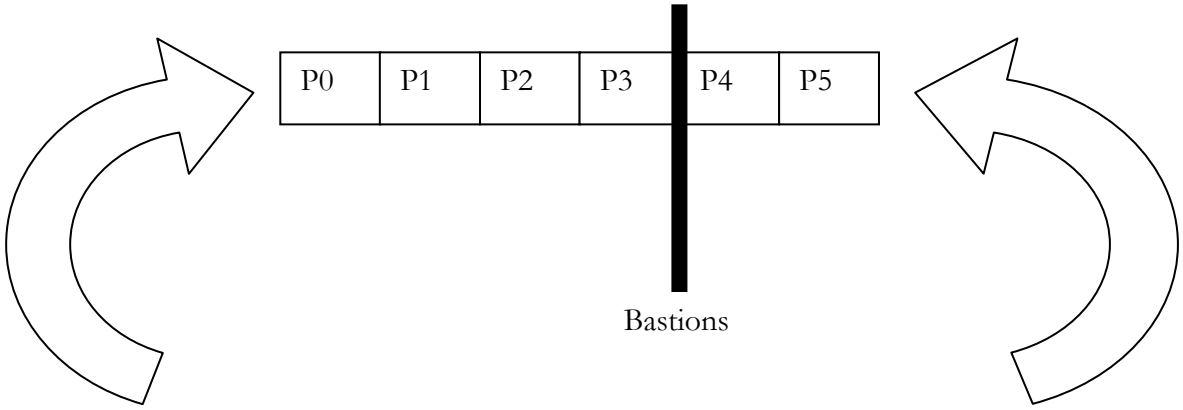
GAME

You are playing a computer game. Initially you have control of all **N** cities in the game. The cities can be seen as a sequence of adjacent squares that are numbered from 0 to N-1 as shown in figure. By the end of each year, you gain some profit (**Pi**) from each city that is still under your control. Note, that the profits associated with each city are not the same since the resources of cites are different. Your kingdom withstood many attacks in the past, but now your enemy is preparing an enormous army hat will crush your whole kingdom. As your defeat is certain, your aim is to have the maximum possible accumulated profit before losing everything.



Each year you choose to build your bastions between 2 adjacent cities that are still under your control. Then, the enemy chooses to attack you either from the west taking control of all cities before the bastions or from the east taking control of all cities after your bastions. By the end of that year, you get the profits associated only with the cities that are still under your control after the attack. Also, by the end of the year, your enemy will succeed to destroy the bastions but he will stop fighting until reinforcements arrive for the next year. In each of the next years, the same scenario happens again taking into consideration only cities that are still under your control.

TASK

This is an interactive task. Your task is to have the maximum possible profit before losing control of all cities. You should achieve this task by carefully choosing where to put your bastions on each year keeping in mind that your enemy is playing optimally concerning his choice of whether attacking from the east or from the west of the bastions (he is trying to minimize your final profit).

LIBRARY

Your program must use a special library to play the game. The library consists of the files: pgamelib.pas (pascal), cgamelib.h and cgamelib.c (C/C++). The library provides the following functionalities:

- procedure initialize ( ) / void initialize ( ) – which must be called only once by your program. This call must be performed before any other calls for any of the following 3 functions.

- function getN: longint / int getN( ) – which returns the number of cities **N** (3 <= **N** <= 2000).
- function getValue(city: longint):longint / int getValue (int city) – which is given a city index (0 <= index < N) returns the profit associated with that city. The profit ranges between 1 and 100,000 inclusive. Calling this function with bad index results in failure for your program for that test case. Note, that city 0 is the leftmost and city N-1 is the rightmost.
- function move(city: longint):longint / int move(int city) – You call this function to specify the index of the city that you wish to have your bastions built right after it. It returns either 1 or 0 indicating whether the enemy will attack from the west (left) or the east (right) respectively. The valid values for the 'city parameter' are only the indices of the cities remaining in your kingdom (except the right-most one of them). Calling this function with bad index results in failure for your program for that test case.

Termination of your program is an automatic process when only one city is under your control. So, your program should keep making moves as long as there are still valid moves.

Your program must not read or write any files, it must not use standard input/output, and it must not try to access any memory outside your program. Violating any of these rules may result in disqualification.

COMPILATION

If your program is written in Pascal, then you must include 'uses pgamelib;' statement in your source code. To compile your program, use the following command: ppc386 -O2 -XS game.pas

If your program is written in C or C++, then you must include ' #include "cgamelib.h" ' statement in your source code. To compile your program, use one of the following commands:
gcc -O2 -static game.c cgamelib.c -lm
g++ -O2 -static game.cpp cgamelib.c –lm

You are provided with two simple programs illustrating usage of the above libraries: cgame.c and pgame.pas. (Please remember, that these programs are not correct solutions)

TESTING

To let you experiment with the library, you are given example opponent libraries: their sources are in pgamelib.pas, cgamelib.h and cgamelib.c files. They implement a simple strategy. When you run your program, it will be playing against these simple opponents. Feel free to modify them, and test your program against a better opponent. However, during the evaluation, your program will be playing against a different opponent that is playing optimally. When you submit your program using the TEST interface it will be compiled with the unmodified example opponent library. The submitted input file will be given to your program standard input. The input file should consist of N+1 lines. The first line contains N and the next N lines contain the sequence of integers specifying the profits of the cities in order from city 0 to city N-1. These values are read by the example opponent library.

If you modify the implementation part of the pgamelib.pas library, please recompile it using the following command: ppc386 -O2 pgamelib.pas
This command produces files pgamelib.o and pgamelib.ppu. These files are needed to compile your program, and should be placed in the directory, where your program is located. Please do not modify the interface part of the pgamelib.pas library.

If you modify the cgamelib.c library, please remember to place it (together with cgamelib.h) in the directory, where your program is located — they are needed to compile it. Please do not modify the cgamelib.h file.

GRADING

If you get the maximum possible profit you receive full credit for this test case. Otherwise, you receive 0 points. Note that it is guaranteed that your opponent is playing optimally.

Also, note that for some tests worth 50 points, **N** will not exceed 500.

SAMPLE INTERACTION

Your Program Calls	Return Value	Comments
initialize( )		
getN( )	5	5 cities
getValue(0)	8	P0 = 8
getValue(1)	6	P1 = 6
getValue(2)	2	P2 = 2
getValue(3)	4	P3 = 4
getValue(4)	2	P4 = 2
move(1)	1	The 2 intervals are [0,1] and [2,4]. Your opponent takes control of [0,1] and you still have control of [2,4], so your score after this turn is P2+P3+P4.
move(2)	0	Note that the allowed moves here were only 2 and 3. After you choose 2, the intervals are [2] and [3,4]. Your opponent takes control of [3,4] and you now have control of only one city so, you add P2 to your score and the game ends.

Your final score with this interaction is 8+2 = 10.