

Stacking containers

The Neptune Cargo Company operates a container storage depot. Its container storage depot accepts containers for storage and subsequent removal.

Containers arrive at the depot for storage every hour on the hour. They stay at the depot for a positive integer number of hours. When a container arrives, its documentation contains the expected time when it will be removed. The first container arrives at time 1. The actual time a container is requested to be removed may ultimately be before or after the expected time by no more than 5 hours.

In this problem, the time in hours is expressed as an increasing positive integer which will not exceed 150.

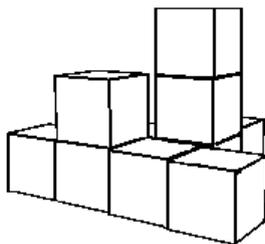
A crane (lifting and moving apparatus) operates above the storage space, moves containers in and out of the storage space, and sometimes rearranges them inside the storage space. The crane may operate in space above the defined storage space.

Task:

You are required to write a program which has a good strategy for accepting, storing and removing containers. A good strategy is one that minimizes the total number of moves that the crane makes.

The depot is a rectangular space. The length (X), width (Y) and height (Z) of the space are made available to the program.

Each container is a 1 x 1 x 1 cube. Containers are stacked on top of other containers or the floor. The crane can only move the top container of a stack.



Moving a container from one location to any other location is always one crane move. All crane moves take place between container arrivals and removals. Crane moves are instantaneous.

When the depot becomes full, your program must refuse to accept any more containers. Your program may become less efficient or unable to continue when the depot is nearly full. Your program may refuse to accept new containers at any time.

Input:

Your program is required to interact with a simulation module which will provide data, and to which your program must submit actions and messages. The depot will be empty when your program starts.

During your program testing, the library will return meaningful values for a small fixed set of test data.

Each container is identified by a unique positive integer.

Your program may call the following functions at any time:

int GetX();
function GetX: integer;
DECLARE FUNCTION GetX CDECL () Returns length of storage space (integer).

int GetY();
function GetY: integer;
DECLARE FUNCTION GetY CDECL () Returns width of storage space (integer).

int GetZ();
function GetZ: integer;
DECLARE FUNCTION GetZ CDECL () Returns height of storage space (integer).

X,Y,Z will not exceed 32.

The following functions provide information on the action sequence (container arrivals and removals). The arrivals take place on the hour, and removal requests are received between hours. Thus, for time-keeping purposes, each arrival marks the passing of one hour.

int GetNextContainer();
function GetNextContainer: integer;
DECLARE FUNCTION GetNextContainer CDECL ()
 Returns a positive integer container number of the next container to be stored or retrieved. If there are no more containers to be stored or retrieved, returns 0, indicating your program should terminate, even if containers are still in the warehouse.

int GetNextAction();
function GetNextAction: integer;
DECLARE FUNCTION GetNextAction CDECL ()
 Returns an integer representing the action to take:
 1 to store a new container, 2 to remove a container.

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

Returns time in hours (since the start) when the container is expected to be removed. This value is for planning purposes for your program; the actual removal request might come at a slightly different time, which will differ by not more than 5 hours. This function only returns a meaningful value when GetNextAction returns 1.

The order in which the above three functions is called does not matter.

Consecutive calls to GetNextContainer, GetNextAction and GetNextStorageTime will always return information about the same container until the container is refused, stored or removed, at which point the above functions will return information about the next container.

Output:

Once your program has found out the information it needs about the next container, use the following functions to manipulate the storage depot:

```
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)
```

Move the container on the top of the stack at x1, y1 to the top of the stack at x2, y2.

Returns 1 if the action is valid, 0 if the action is illegal (i.e. impossible).

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

Refuse to accept the incoming container.

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

Store the incoming container at the top of the stack at position 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)
```

Remove the container on the top of the stack at x, y from the depot.

If your program cannot carry out the required action, it should terminate.

Illegal moves are ignored by the library, and have no effect on the simulation state or scoring.

Your program is NOT required to write any output to a file. The library with which your program interacts will write a log file of actions. This file is used for evaluation.

Sequencing:

Your program should get information about the next container request. It should then move containers with the crane if desired and subsequently store, remove or refuse the action request.

Library:

A library called *StackLib* is provided which you must link to your code.

The standard C and C++ libraries contain this library and will automatically be linked to your program when you include the appropriate header file.

If you are using QuickBasic you must include the library by typing

```
QB /L STACKLIB
```

Sample source code files are present in the task directory named TESTSTK.BAS, TESTSTK.PAS, TESTSTK.CPP, and TESTSTK.C.

Scoring:

The program will be tested with several data sets and for each data set, its performance will be scored against the most efficient solution known to us, using the following indicators:

- Total number of crane moves by your program.
- A penalty of 5 moves is imposed for every container refused.
- A penalty of 5 moves is imposed for each container not stored and removed (i.e. if the program terminates normally before the entire operation is complete).
- The total score will be calculated relative to the best known solution.
- If the program makes more than twice the number of operations necessary, it scores 0.
- The minimum score is 0%, and the maximum score is 100%.

