Tema 1 - Robot Planning

Responsabili:

- Teodora Șerbănescu [mailto:mailto:teo.serbanescu16@gmail.com]
- George Popescu [mailto:mailto:george.apopescu97@gmail.com]
- Data publicării: 14 martie 2018
- Deadline: 4 aprilie 2018

Modificări și actualizări

- **17 martie**
  - adaugare biblioteci <sstream> si <cstring>
  - corectat formatul output-ului din exemplu :)
- **19 martie**
  - adaugare restrictie de genericitate a structurilor de date
- **21 martie**
  - modificare comportament undo pentru execute
    - comanda executata anterior trebuie adaugata **mereu** la inceputul cozii
- **22 martie**
  - modificare checker
    - adaugare fisier .git pentru checkerul de coding style

Obiective

- Aprofundarea cunoștințelor în utilizarea limbajului C++
- Implementarea și utilizarea structurilor de date listă, stivă și double-ended queue(deque) (generice).

Intro

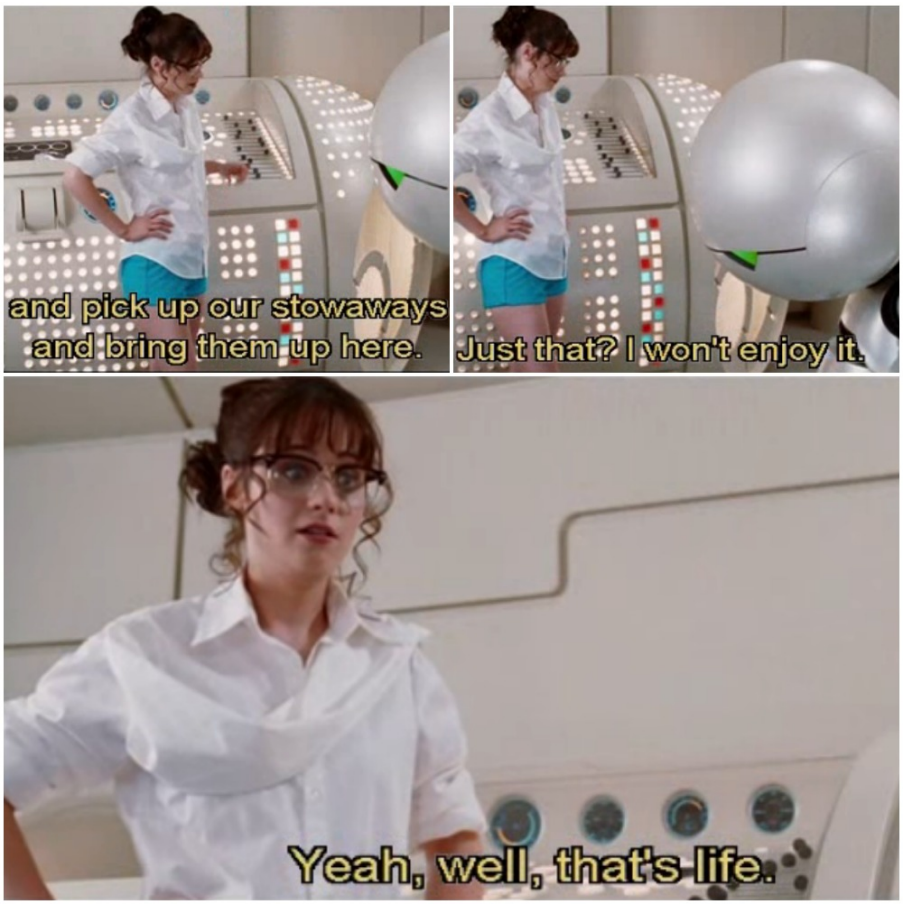
Într-un depozit, definit printr-o configurație map[LIN][COL], se află N roboți responsabili de mutarea cutiilor. map[x][y] conține numărul de cutii din celula respectivă. Cineva controlează toți roboții, dându-le comenzi astfel încât ei să strângă/livreze cutiile din anumite celule.

Comenzile sunt de tipul:

- **ADD_GET_BOX** RobotId x y NrBoxes Priority
- **ADD_DROP_BOX** RobotId x y NrBoxes Priority
- **EXECUTE** RobotId
- **PRINT_COMMANDS** RobotId
- **LAST_EXECUTED_COMMAND**
- **UNDO**
- **HOW_MUCH_TIME**
- **HOW_MANY_BOXES** RobotId

Comenzile de tipul “GET” și “DROP” sunt reținute de roboți într-o coadă, executând câte una din ele atunci când se apelează “EXECUTE” pentru robotul respectiv, mai exact prima.

Important: Robotul are acces rapid la prima și ultima comandă din coada sa de execuție, putând să adauge la ambele capete comenzile primite.



Descrierea comenzilor:

1.**ADD_GET_BOX** RobotId x y NrBoxes Priority

ADD_DROP_BOX RobotId x y NrBoxes Priority

Adaugă în coada robotului RobotId (CommandType, x, y, NrBoxes), în funcție de Priority.

CommandType poate fi "GET" sau "DROP".

(x, y) este celula din hartă de unde trebuie sa strangă/unde trebuie să livreze cutii (se garantează că x și y sunt valizi).

NrBoxes este numărul de cutii pe care trebuie să le strangă/livreze.

Priority poate fi 0 sau 1, cu semnificația:

- Dacă Priority == 0, (CommandType, x, y, NrBoxes) e adăugat la începutul cozii de comenzi.
- Dacă Priority == 1, (CommandType, x, y, NrBoxes) e adăugat la sfârșitul cozii de comenzi.

2. **EXECUTE** RobotId

Execută prima comandă din coada lui RobotId.

Dacă prima comandă este de tipul "GET" robotul strânge firstNrBoxes cutii din celula map[firstX][firstY]. Dacă celula are mai puțin de firstNrBoxes cutii, robotul strânge câte cutii sunt în ea.

Dacă este de tipul "DROP" robotul adaugă în celula map[x][y] firstNrBoxes cutii. Dacă robotul are mai puțin de firstNrBoxes cutii, va adăuga câte cutii are.

Dacă coada robotului RobotId nu conține nicio comandă, se va afișa "No command to execute".

3. **PRINT_COMMANDS** RobotId

Afișează comenzile puse în coada robotului RobotId, în ordinea corectă, de la prima la ultima. Afișarea este de forma:

```
RobotId: CommandType0 x0 y0 NrBoxes0; CommandType1 x1 y1 NrBoxes1; CommandType2 x2 y2 NrBoxes2; …….. ; LastCommandType lastX lastY LastNrBoxes
```

Dacă robotul nu are nicio comandă în coadă, se va afișa "No command found".

4. **LAST_EXECUTED_COMMAND**

Va afișa un output de forma:

```
RobotId: CommandType x y NrBoxes
```

NrBoxes este numărul efectiv de cutii pe care le-a ridicat/lăsat robotul când a executat comanda. (e.g. Dacă comanda cerea să ridice 10 cutii, dar în celula respectivă erau doar 5, NrBoxes este 5).

Dacă o comandă(de tipul ADD_GET_BOX, ADD_DROP_BOX, EXECUTE) a fost executată, dar după s-a aplicat operația UNDO, această nu va mai fi considerată ca executată.

Dacă nu a fost executată nicio comandă, se va afișa "No command was executed".

5. **UNDO**

Anulează efectul ultimei comenzi de tipul ADD sau EXECUTE dată de om.

- Dacă ultima comandă a fost ADD_GET_BOX sau ADD_DROP_BOX, operația de undo va elimina din coada robotului respectiv comanda adăugată.
- Dacă ultima comandă a fost EXECUTE, operația de undo va pune LAST_EXECUTED_COMMAND la loc în coadă de comenzi a robotului său.

Precizări

- După ce se face undo la o operație de tipul execute, ultima comandă executată va fi adăugată la loc în coada robotului corespunzător, la începutul ei, iar NrBoxes va fi egal cu numărul efectiv de cutii strânse/livrate de acesta.

- Cutiile vor fi puse la loc în hartă, în cazul anulării execuției unui GET, respectiv luate din harta și puse la loc în inventarul robotului, în cazul anulării execuției unui DROP.

- **NU** se iau în considerare pentru undo restul comenzilor(nu se șterge din fișierul de output răspunsul pentru acele cerințe).

- Dacă între comenzile de tipul ADD/EXECUTE au fost date și alte comenzi care nu influențează roboții, acestea din urmă nu se iau în calcul, căutându-se ultima comandă asupra căreia se poate aplica UNDO.

- Dacă undo nu are o comandă de tip ADD sau EXECUTE pe care să o anuleze, se afișează mesajul "No History".

6. **HOW_MUCH_TIME**

Afișează un întreg reprezentând cât timp a stat LAST_EXECUTED_COMMAND în coada robotului său, înainte să fie executată.

Timpul se măsoară în numărul de comenzi date de om de la introducerea LAST_EXECUTED_COMMAND în coadă, până la execuția sa. Se iau în calcul toate comenzile, indiferent de tipul lor sau de id-ul roboților cărora le sunt date.

Dacă o comandă este scoasă din coadă și executată, dar după s-a apelat UNDO, comanda este introdusă din nou în coadă, iar timpul de inserare va fi considerat cel de la inserarea curentă (după undo), nu cel de la prima inserare.

Dacă nu a fost executată nicio comandă se va afișa "No command was executed".

7. **HOW_MANY_BOXES** RobotId

Afișează un întreg reprezentând câte cutii cară în acel moment de timp robotul cu id-ul RobotId.

Atenție

Toate afișările din output vor fi prefixate de tipul comenzii care le-a generat, astfel:

```
Tip_Comanda: Output
```

De exemplu:

```
EXECUTE: No command to execute
UNDO: No history
PRINT_COMMANDS: 0: GET 1 2 3; DROP 0 1 1
HOW_MUCH_TIME: 3
HOW_MANY_BOXES: 4
etc.
```

Cerință

Dându-se mai multe operații, simulați comportamentul roboților și afișați detaliile, conform cu descrierea de mai sus a comenzilor.

Atenție: Nu se da numărul de operații.

Implementare

Structuri de date obligatorii

Deque - implementat folosind DoublyLinkedList.

Stack - implementat folosind ResizableArray.

Atenție! Scopul acestei teme este să învățați să implementati și folosiți aceste structuri de date, nerespectarea acestor cerințe vă va aduce 0p pe tema.

Atenție! Structurile de date trebuie sa fie implementate generic.

Date de intrare

Datele de intrare sunt în fișierul "robots.in".

Pe prima linie: N LIN COL

Urmează LIN linii cu câte COL elemente, reprezentând ce se află inițial în hartă.

În continuare, sunt date pe câte o linie comenzile. Nu se știe numărul lor, trebuie citit până la finalul fișierului.

Date de ieșire

Fiecare comandă (dintre cele care afișează ceva) va afișa mesajul corespunzător pe câte o linie.

Datele de ieșire se vor scrie în fișierul "robots.out".

Teste publice

Checker Tema1

Exemplu

Input (robots.in)

```
2 3 3
1 2 3
2 3 1
2 5 3
ADD_GET_BOX 0 1 2 3 0
ADD_DROP_BOX 0 0 1 1 1
ADD_GET_BOX 1 2 2 2 0
UNDO
EXECUTE 1
PRINT_COMMANDS 0
EXECUTE 0
LAST_EXECUTED_COMMAND
HOW_MUCH_TIME
PRINT_COMMANDS 0
HOW_MANY_BOXES 0
UNDO
HOW_MUCH_TIME
HOW_MANY_BOXES 0
EXECUTE 0
HOW_MUCH_TIME
```

Output (robots.out)

```
EXECUTE: No command to execute
PRINT_COMMANDS: 0: GET 1 2 3; DROP 0 1 1
LAST_EXECUTED_COMMAND: 0: GET 1 2 1
HOW_MUCH_TIME: 6
PRINT_COMMANDS: 0: DROP 0 1 1
HOW_MANY_BOXES: 1
HOW_MUCH_TIME: No command was executed
HOW_MANY_BOXES: 0
HOW_MUCH_TIME: 3
```

Reguli pentru trimitere

Temele vor trebui trimise pe vmchecker [https://elf.cs.pub.ro/vmchecker/ui/#SD]. **Atenție!** Temele trebuie trimise în secțiunea **Structuri de Date (CA)**.

Arhiva trebuie să conțină:

- surse
- fișier Makefile cu două reguli:
  - regula **build**: în urma căreia se generează un executabil numit **tema1**
  - regula **clean** care şterge executabilul
- fisier **README** care să conțină detalii despre implementarea temei

Punctaj

1. 80p teste
2. **Fiecare** test este verificat cu valgrind. Dacă un test are memory leaks, nu va fi punctat.
3. 20p README + comentarii/claritate cod (ATENȚIE! Fișierul README trebuie făcut explicit, cât să se înțeleagă ce ați făcut în sursă, dar fără comentarii inutile și detalii inutile)
4. Se acordă 20% din punctajul obținut pe teste, ca bonus pentru coding-style. De exemplu, pentru o temă care obține maxim pe teste, se pot obține 20p bonus dacă nu aveți erori de coding style. Pentru o temă ce trece 8 teste din 10, se pot obține 16p dacă nu aveți erori de coding style.
5. O temă care obține 0p pe vmchecker este punctată cu 0.

Nu copiați! Toate soluțiile vor fi verificate folosind o unealtă de detectare a plagiatului. În cazul detectării unui astfel de caz, atât plagiatorul cât și autorul original (nu contează cine care e) vor primi punctaj 0 pe **toate temele**!

De aceea, vă sfătuim să nu vă lăsați rezolvări ale temelor pe calculatoare partajate (la laborator etc), pe mail/liste de discuții/grupuri etc.

FAQ

Q: Se poate folosi STL?

A: Nu puteți folosi structurile gata implementate în STL. Obiectivul principal al cursului de structuri de date este acela ca voi să implementați structurile respective.

Q: Ce biblioteci putem folosi?

A: Aveți voie să folosiți următoarele biblioteci C/C++: <iostream>, <fstream>, <cstdio>, <string>, <sstream>, <tuple>, <cstring>. Va recomandăm să folosiți <tuple>, pentru a va crea mai ușor structuri în care doar trebuie să stocați date. (http://www.cplusplus.com/reference/tuple/tuple/?kw=tuple [http://www.cplusplus.com/reference/tuple/tuple/?kw=tuple]) Orice structura de date pe care nu aveți voie să o folosiți, dar o folosiți, vă va aduce 0p pe tema.