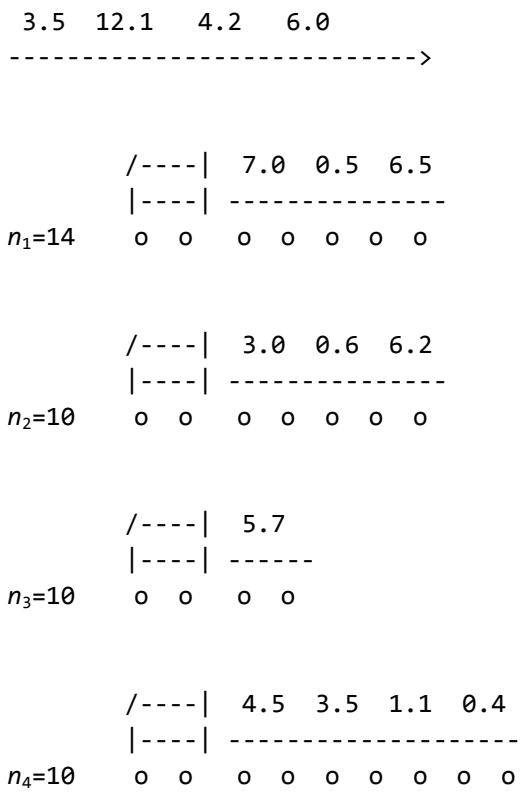
Zadanie 1.

Wyobraź sobie taśmociąg, którym podawane są kolejno przedmioty p_1, p_2 ,..., p_k.

Każdy przedmiot p_i ma określoną wagę w_i. Przedmioty, w kolejności dostarczania przez taśmociąg, należy pakować do samochodów o ograniczonej ładowności n_i , która może być różna dla każdego samochodu. Przyjmij, że ilość przedmiotów, które można załadować do samochodu jest nieograniczona; kontroluj jedynie czy suma wag nie przekracza ładowności. W sytuacji, gdy dołożenie kolejnego przedmiotu spowodowałoby przekroczenie ładowności, należy podstawić kolejny pusty samochód. Proces pakowania prowadzony jest dopóki taśmociąg dostarcza kolejne przedmioty.

Napisz program symulujący powyższy proces. Taśmociąg zaimplementuj, jako kolejkę STL (queue) o wartościach elementów generowanych losowo. Samochody powinny być zaimplementowane, jako stosy tworzone dynamicznie (w miarę potrzeb). Zapamiętuj samochody w tablicy realizowanej, jako wektor STL (vector). Własną klasę stosu zdefiniuj korzystając ze stosu STL (stack), rozszerzając ją o składowe przechowujące pojemność i aktualną zajętość stosu oraz odpowiednie metody pozwalające na operowanie na nich.



Zadanie 2.

Napisz program przeszukujący graf wszerz metodą BFS. Jako strukturę sterującą kolejnością przeszukiwania wierzchołków wykorzystaj kolejkę (queue) z biblioteki STL. Graf reprezentuj, jako wektor STL (vector), którego każdy element jest wektorem sąsiadów odpowiedniego wierzchołka.

Zadanie 3.

Napisz program przeszukujący graf w głąb metodą DFS. Jako strukturę sterującą kolejnością przeszukiwania wierzchołków wykorzystaj stos (stack) z biblioteki STL. Graf reprezentuj, jako wektor STL (vector), którego każdy element wektorem sąsiadów odpowiedniego wierzchołka.

Zadanie 4.

Napisz program symulujący karcianą grę w wojnę.

Opis zasad znajdziesz opd adresem

http://pl.wikipedia.org/wiki/Wojna_(gra_karciana)

Talię kart reprezentuj, jako wektor (vector) STL, którego elementami są karty. Do reprezentacji zbioru kart posiadanych przez każdego z graczy wykorzystaj kolejkę (queue) STL. Gracze wykładają karty na stół, gdzie są one porównywane. Zdefiniuj strukturę danych do reprezentacji stołu wykorzystując w jej definicji elementy biblioteki STL.

Zadanie 5.

Napisz program pozwalający układać pasjans Klondike (reguły patrz gra Pasjans w systemie Windows). Program powinien wykorzystywać kontenery vector, stack oraz queue biblioteki STL. Można zastosować wektory (1-7), wektor czterech stosów (8-11) oraz kolejkę (0). Konieczne jest zaimplementowanie reguł sprawdzających poprawność ruchów.

```
      0                            8      9     10    11
-------------------------------------------------------
   Kh 9s Kd                      Ah     2c     As     0
-------------------------------------------------------
|      |  10h  |  ===  |  ===  |  ===  |  ===  |  ===  |
|      |   9c  |  ===  |  ===  |  ===  |  ===  |  ===  |
|      |       |   3s  |   Js  |  ===  |  ===  |  ===  |
|      |       |       |       |   Jd  |  ===  |  ===  |
|      |       |       |       |  10c  |  ===  |  ===  |
|      |       |       |       |   9d  |   7h  |  ===  |
|      |       |       |       |   8c  |       |   5c  |
|      |       |       |       |       |       |   4d  |
|      |       |       |       |       |       |   3c  |
|      |       |       |       |       |       |   2h  |
   1       2       3       4       5       6       7
```

Przykładowe ruchy

>> 0 Kd 1

>> 2 10h 4

>> 6 7h 5

>> 7 2h 9

>> 0

h - hearts - kier

s - spades - pik

d - diamond - karo

c - club - trefl

Zadanie 6.

Napisz program, w którym użytkownik jest kontrolerem bocznicy kolejowej.

Sterowanie rozjazdami odbywa się poprzez klawisze 2-5, zwalnianie wagonu klawiszem ‘w’.

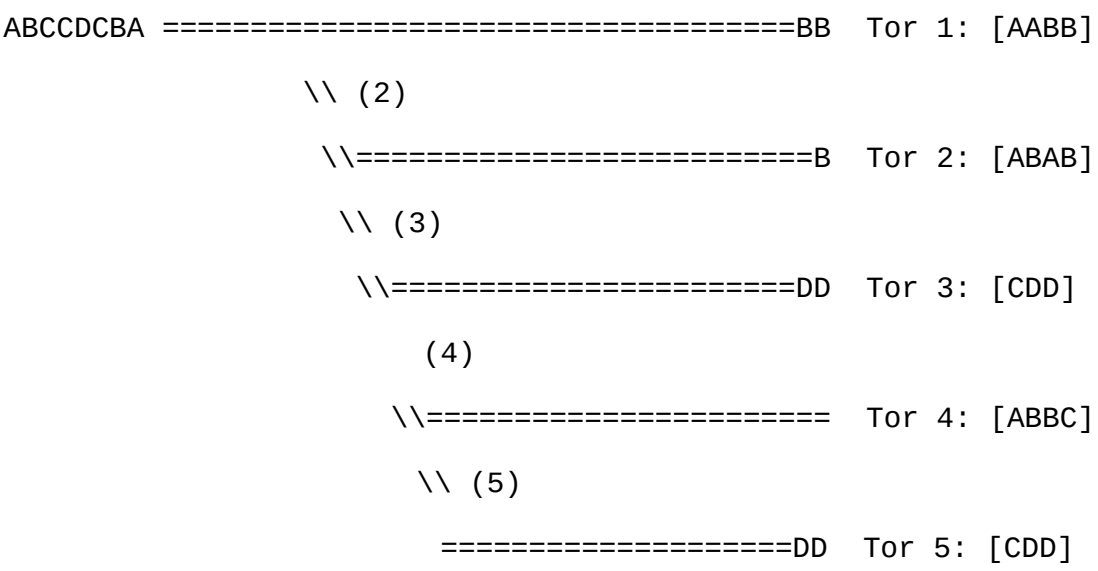
Zadaniem kontrolera jest odpowiednie rozmieszczenie wagonów - zgodnie z podpowiedziami obok każdego toru.

Po skompletowaniu prawidłowego składu tor jest zwalniany a użytkownik otrzymuje 10 pkt.

W przypadku jeśli nie jest możliwa prawidłowa kompletacja (użytkownik się pomylił) odejmowanych jest 10 pkt.

Użytkownik wygrywa, gdy osiągnie 50 pkt, przegrywa jeśli liczba punktów będzie mniejsza od zera.

Program powinien wykorzystywać kontenery biblioteki STL.



Zadanie 7.

Napisz program symulujący wieżę kontroli lotów. Kontroler ma za zadanie przeprowadzić samoloty (pojawiające się w losowych odstępach czasu) z lewej do prawej lub z prawej do lewej strony ekranu tak, aby nie nastąpiła kolizja.

Ruch samolotów odbywa się turowo, jeden znak na turę.

Legenda:

(A3)/ - samolot A, kierunek lotu z lewej do prawej, faza wznoszenia (jeszcze 3 pola do góry)

=(E0) – samolot E, kierunek lotu z lewej do prawej, lot stabilny

(D1)\ - samolot D, kierunek lotu z lewej do prawej, faza opadania (jeszcze 1 pole w dół)

Komendy:

<spacja> - następna tura

<znak samolotu> / <liczba 1-9> - nakaz wzniesienia się o podaną liczbę pól

<znak samolotu> \ <liczba 1-9> - nakaz opadania o podaną liczbę pól

<znak samolotu> c – anulowanie rozkazu

Zmiana wysokości (opadanie lub wznoszenie) następuje ze zwłoką jednej tury.

W każdej chwili samoloty muszą znajdować się co najmniej w odległości dwóch znaków od siebie.

Program powinien wykorzystywać kontenery biblioteki STL.

Przykładowy widok ekranu (60x10 znaków), samoloty oczekujące pokazane są przy obramowaniu:

```
  ============================================================
  |(A1)/                                                     |
  |                                =(E0)                     |
  |                                                          |
 B|                                                          |
  |                                                          |
  |                                                          |C
  |                             (D2)\                        |
  |                                                          |
  ============================================================
```

Zadanie 8.

Napisz program pozwalający układać pasjans Freecell (reguły: http://pl.wikipedia.org/wiki/FreeCell).

Program powinien wykorzystywać kontenery biblioteki STL.

Konieczne jest zaimplementowanie reguł sprawdzających poprawność ruchów.

Zadanie 9.

Napisz kalkulator wykonujący obliczenia z użyciem odwrotnej notacji polskiej.

Użytkownik wprowadza używając notacji zwykłej, np.:

(3 + 7) * 2 - 6

Program powinien wykorzystywać kontenery biblioteki STL.

Przykład 1:

Notacja zwykła:	Odwrotna notacja polska
2 + 3	2 3 +

Przykład 2:

Notacja zwykła:	Odwrotna notacja polska
(3 + 7) * 2 - 6	3 7 + 2 * 6 -
10 * 2 - 6	10 2 * 6 -
20 - 6	20 6 -
14	14

Zadanie 10.

Gra karciana. W rozgrywce bierze udział dwóch graczy: gracz 1 (człowiek) i gracz 2 (komputer). Każdy z graczy otrzymuje po 7 kart ze standardowej talii składającej się z 52 kart.

Zasady gry:

Każdy z graczy w swojej turze pyta przeciwnika o karty danej wysokości, np.: gracz 1 pyta o ósemki.

Jeśli gracz 2 posiada ósemki na swoim ręku przekazuje je graczowi 1, a gracz 1 pyta dalej.

Jeśli gracz 2 nie posiada kart, o które pyta gracz 1, gracz 1 otrzymuje stosowną odpowiedź, losuje kartę z talii i kończy turę.

Uwaga: dany gracz może pytać tylko o te karty, z których przynajmniej jedną ma w swoim ręku.

Jeśli dany gracz zbierze wszystkie karty danej wysokości (np. 4 ósemki), formuje z nich „stos".

Gra kończy się w chwili, gdy wszystkie karty z talii zostaną wykorzystane.

Wygrywa gracz z większą liczbą „stosów".

Program powinien wykorzystywać kontenery biblioteki STL.

Fragment przykładowej rozgrywki (T – trefl, K – kier, k – karo, P – pik):

```
Stosy komputera: brak
Twoje karty: 2T 4K 8k JT DK KK AP
Pytaj o: 8
Komputer przekazuje Ci 8K 8P 8T

Stosy komputera: brak
Twoje karty: 2T 4K JT DK KK AP + stos 8
Pytaj o: 4
Komputer: nie mam!
Wylosowales 10P

Stosy komputera: brak
Twoje karty: 2T 4K 10P JT DK KK AP + stos 8
Komputer pyta o 10
Przekazujesz komputerowi 10P

Stosy komputera: 10
Twoje karty: 2T 4K JT DK KK AP + stos 8
Komputer pyta o ...
```

Zadanie 11.

Napisz program pozwalający grać na konsoli **w statyczny** wariant gry typu arkanoid:

- plansza składa się z w wierszy oraz *k* kolumn zawierających początkowo *b* bloczków każda,
- jeden ruch gracza (strzał) polega na wyborze jednej spośród kolumn, w której zostanie zlikwidowany jeden bloczek,
- co *r*+*rnd* ruchów do wszystkich kolumn dodawany jest jeden bloczek zmniejszając przestrzeń dla gracza. Parametr *rnd* jest wartością losową z przedziału [0,*n*),
- gra kończy się gdy jedna z kolumn dojdzie do wiersza, w którym znajduje się gracz.

Szczegółowe reguły ustal samodzielnie.

W niektórych bloczkach możesz ukryć bonusy :: ++ itp.

Program powinien pozwalać na dobór parametrów *k*, *b*, *r*, *n*.

Do reprezentacji obiektów zastosuj wektor STL (vector) zawierający *k* kolejek STL (queue).

Przykład: Wygląd ekranu podczas działania programu

```
----------------------------------------
####  ####  ####  ####  ####  ####  ####
####  ####  ####  #::#  ####  ####  #++#
####  ####        ####  ####        ####
####              ####              ####
####

                   =[]=
----------------------------------------
```