ამ დავალებას რამოდენიმე მიზანი აქვს:

1. მიიღოთ წინასწარ გამზადებულ კლასებთან მუშაობის უფრო მეტი გამოცდილება. “CS106” ბიბლიოთეკაში არსებული ობიექტების გამოყენებით ადვილად შეძლებთ, ბევრი რთული ოპერაციის შესრულებას.

2. გაეცნოთ აბსტრაქციის ცნებას, როგორც მონაცემთა მართვის მექანიზმს და ყურადღება მიაქციოთ იმ ფაქტს, რომ აბსტრაქცია გვაძლევს საშუალებას გამოვიყენოთ ფუნქციონალი, რომლის კონკრეტული რეალიზაციაც ჩვენთვის დამალულია.

3. გაეცნოთ C++ ტემპლეიტების გამოყენებას.

4. გაეცნოთ და პრაქტიკაში გამოიყენოთ კლასიკური მონაცემთა სტრუქტურები, როგორიცაა სტეკი, სია, ვექტორი, მეფი და სიმრავლე.

**ამოცანა 1. Word ladders (სიტყვების ბილიკი)**

**სიტყვების ბილიკი** შედგება მიმდევრობით ჩაწერილი სიტყვებისგან სადაც ყოველი ორი მეზობელი სიტყვა მიიღება ერთერთში მხოლოდ 1 სიმბოლოს შეცვლით.   
მაგალითისთვის ქვემოთ მოცემულია სიტყვების ბილიკი რომელსაც “code” დან მივყავართ “data” სიტყვამდე.

**code core care dare date data**

ზემოთ მოცემული ბილიკი არ არის უმოკლესი, შესაძლებელია ერთი ბიჯით ნაკლები ბილიკის საშუალებითაც მივიდეთ “code” დან “data” მდე.

**code** **cade** **cate** **date** **data**

თქვენი დავალებაა იპოვოთ უმოკლესი **სიტყვების ბილიკი**, მოცემული ორი სიტყვისთვის.

სურათზე ნაჩვენებია პროგრამის გაშვების მაგალითი:



ალგორითმის მონახაზი

უმოკლესი მანძილის პოვნა ძალიან გავრცელებული პრობლემაა ბევრ სფეროში, მაგალითად ინფორმაციის გადაცემისას სხვადასხვა როუტერის გავლით, რობოტის გადაადგილების დაგეგმვისას, ახლო სოციალურ ქსელში კავშირების დადგენისას, გენის მუტაციების შედარებისას და სხვა. სიტყვების ბილიკის პოვნა მსგავსი ტიპის ამოცანამდე დაიყვანება.

უმოკლესი მანძილის პოვნისთვის ხშირად გამოიყენება სიგანეში ძებნის ალგორითმი, რომელიც პირველ რიგში განიხილავს ერთ ბიჯით დაშორებულ პოზიციებს, მერე ორი ბიჯით და ასე შემდეგ, მანამ სანამ არ იპოვის ამონახსნს.

ჩვენს შემთხვევაში სიგანეში ძებნის სტრატეგია პირველ რიგში განიხილავს სიტყვებს რომლებიც მიიღება საწყის სიტყვაში მხოლოდ ერთი სიმბოლოს შეცვლით, თუ რომელიმე ცვლილების გზით მივიღებთ დანიშნულების სიტყვას მაშინ ამონახსნი ნაპოვნია, წინააღმდეგ შემთხვევაში ალგორითმი მომდევნო ბიჯზე განიხილავს ყველა ბილიკს რომელიც ორი ნაბიჯით არის დაშორებული საწყისი სიტყვიდან და ასე შემდეგ, მანამ სანამ არ მივიღებთ დანიშნულების სიტყვას.

სიგანეში ძებნის ალგორითმის რეალიზაციის დროს ხშირად იყენებენ რიგის სტრუქტურას, რომელშიც ინახება ისაწყისი წერტილიდან მიღწეული პოზიციები.  
ჩვენს შემთხვევაში რიგში შევინახავთ საწყისი სიტყვიდან გამოსულ ბილიკებს, ჯერ განვიხილავთ ერთი სიგრძის ბილიკებს, შემდეგ ორი და ასე შემდეგ მანამ სანამ არ მივაღწევთ დანიშნულების სიტყვამდე.

იხილეთ ფსევდო კოდი რომელიც დეტალურად აღწერს ალგორითმს.

*შექმენით ცარიელი რიგი.  
ჩაამატეთ რიგში საწყისი სიტყვა.*while (*რიგი არ არის ცარიელი*) { *ამოვიღოთ რიგიდან ბილიკი.* if (*რიგიდან ამოღებული ბილიკის ბოლო სიტყვა ემთხვევა საბოლოო სიტყვას*) { *დააბრუნე რიგიდან ამოღებული ბილიკი როგორც ამონახსნი* }

for ( *თითოეული სიტყვისთვის ინგლისური ლექსიკონიდან, რომელიც მიიღება რიგიდან ამოღებულ ბილიკის ბოლო სიტყვაში ერთი ასოს შეცვლით*) {if (*თუ სიტყვა ჯერ არ გამოგვიყენებია რიგში არსებულ ბილიკებში*) { *გააკეთეთ რიგიდან ამოღებული ბილიკის კოპია  
 დაამატეთ კოპია ბილიკს სიტყვა ლექსიკონიდან  
 დაამატეთ ახალი ბილიკი რიგში* }

}} *ბილიკი საერთოდ არ არსებობს*

ყურადღება მიაქციეთ პრიობას if რომელიც საჭირა ორი მიზეზის გამო:  
 if (*თუ სიტყვა ჯერ არ გამოგვიყენებია რიგში არსებულ ბილიკებში*)

1. იმისათვის რომ თავიდან ავირიდოთ რიგში არა საჭირო ბილიკების შენახვა, თუ სიტყვა აქამდე განვიხილე, ესე იგი ამ სიტყვამდე უმოკლესი გზა უკვე ნაპოვნია, შესავამისად დამატებითი გზებს რიგში ჩაყრა არაფერს არ მოგვცემს.
2. იმ შემთხვევაში როცა მოცემულ საწყის და საბოლოო სიტყვებს შორის არ არსებობს ბილიკი. პირობის გარეშე ალგორითმი შესაძლოა ჩაიციკლოს.  
   მაგ: **code core  code core  code core  code ...**და ასე უსასრულოდ

ამ პირობის შემოწმებისთვის თქვენ დაგჭირდებათ ყველა იმ სიტყვის შენახვა რომელიც ბილიკების პოვნის დროს გხვდებათ, შესაძლოა Set ან Lexicon სტრუქტურა გამოგადგეთ.

იმპლემენტაცია

ჩვენი რჩევა იქნება დავალება შეასრულოთ შემდეგი გეგმის მიხედვით:

1. გაეცანით Demo პროგრამას, იმისათვის რომ კარგად წარმოიდგინოთ როგორი უნდა იყოს თქვენი პროგრამა მომხმარებლის გადმოსახედიდან.
2. ყურადღებით წაიკითხეთ ყველა იმ კლასის დოკუმენტაცია რომელიც დავალების შესრულების დროს დაგჭირდებათ **Vector**, **Queue**, and **Lexicon \ Set.**  კოდის წერის დაწყებამდე აუცილებელია კარგად გესმოდეს თითოეული სტრუქტურის შინაარსი.
3. კარგად გაიაზრეთ დავალებაში აღწერილი ალგორითმი და მონაცემთა სტრუქტურების დიზაინი.
4. დაწერეთ რამდენიმე სავარჯიშო პროგრამა Lexicon კლასის გამოყენებით. მაგალითად დაწერეთ პროგრამა რომელიც კონსოლიდან შეყვანილი სიტყვისთვის ბეჭდავს, სიტყვებს ინგლისური ლექსიკონიდან, რომლებიც მიიღებიან საწყის სიტყვაში მხოლოდ ერთი სიმბოლოს შეცვლით. ინგლისური ლექსიკონისთვის გამოიყენეთ **EnglishWords.dat** ფაილი.
5. დაწერეთ სიგანეში ძებნის ალგორითმი. აუცილებლად გატესტედ პროგრამა, შეგიძლიათ თქვენით შექმნათ რაიმე პატარა ლექსიკონი, რომლისთვისაც ადვილად მიხვდებით სწორად იმუშავა თუ არა ალგორითმა.

**ამოცანა 1. Random Generator**

ბოლო რამოდენიმე ათწლეულის მანძილზე კომპიუტერებმა რევოლუცია მოახდინეს სტუდენტების ცხოვრებაში. მიუხედავად უამრავი გასართობი პროგრამებისა კომპიუტერებმა საგრძნობლად გაუმჯობესეს სტუდენტის პროდუქტიულობა. თუმცა სტუდენტური შრომის ზოგიერთი მნიშვნელოვანი ნაწილი ამ პროგრესის მიღმა დარჩა. ერთერთი ასეთი ნაწილია გარკვეული დოკუმენტების შევსება. მაგალითად, სხვადასხვა განცხადებები, დისერტაციები, სარეკომენდაციო წერილები მაღალფარდოვანი სიტყვებით და ასე თუ ისე აზრიანი სიტყვების რახა–რუხით. ალბათ თქვენც იცით რომ ისედაც გადაღლილ პროფესორებს არ აქვთ იმდენი დრო რომ დაწვრილებით გაეცნონ ასეთ დოკუმენტებს.

ასეთი პრობლემატური საკითხის მოსაგვარებლად იდეალურია, შემთხვევითი ტექსტის გენერატორი, რომელიც დააგენერირებს აზრიან ტექსტს, უკვე არსებული ტექსტის ანალიზის საფუძველზე. მაგალითად თუ თქვენ ხვალინდელი დღისთვის გასამზადებელი გაქვთ 10 გვერდიანი დოკუმენტი და ვერ ასწრებთ, 8 გვერდის დაწერის შემდეგ თამამად შეგიძლიათ შემთხვევითი ტექსტის გენერატორს გადასცეთ თქვენი ტექსტი და ის დაგიგენერირებთ დანარჩენ ორ გვერდს.

**როგორ ?**

შემთხვევითი ტექსტის გენერატორის იდეა კლოდ შენონს(Claude Shannon) ეკუთვნის. მისი ცნობილი სტატია აღწერს მარკოვის(Markov) მოდელს ინგლისური ტექსტისთვის. მარკოვის მოდელი სტატისტიკური მოდელია, რომელიც აღწერს სისტემის მომავალ მდგომარეობას ახლანდელი მდგომარეობის ანალიზის საფუძველზე. მარკოვის მოდელი ფართოდ გამოიყენება ხელნაწერის და ხმის ამომცნობ სისტემებში, მანქანური სწავლების სისტემებში, ბიოინფორმატიკაში, უფრო მეტიც Google–ის ვებ გვერდების სორტირების ალგორითმიც კი შეიცავს მარკოვის მოდელის კომპონენტებს. ინგლისური ტექსტისთვის მარკოვის მოდელი, სიმბოლოების მიდევრობისთის ითვლის შემდეგი სიმბოლოს ამოსვლის ალბათობას. რათქმაუნდა სიმბოლოების მიმდევრობა ტექსტში არ არის ასოების შემთხვევითი განლაგება, მარკოვის მოდიელი იძლევა საშუალებას ტექსტის ანალიზის საფუძველზე აიგოს ბაზისური პატერნები და შემდეგ ამ პატერნების საშუალებით დააგენერიროს ახალი ტექსტი რომელიც აკმაყოფილებს მოდელს.

შემთხვევითი ტექსტის გენერატორის ყველაზე მარტივი ვარიანტია აბსლუტურად შემთხვევთად სიმბოლოების დაგენერირება. ამ შემთხვევაში რა თქმა უნდა არანაირ მოდელთან არ გვაქვს საქმე.

**No model** aowk fh4.s8an zp[q;1k ss4o2mai/

მცირედი გაუმჯობესება იქნება, თუ გავითვალისწინებთ იმ ფაქტს, რომ ინგლისურ ტექსტში ყველა სიმბოლო არ გვხვდება თანაბარი სიხშირით. შეგვიძლია დავთვალოთ სიმბოლოების სიხშირე და ამ ინფორმაციის გათვალისწინებით გამოვითვალოთ კონკრეტული სიმბოლოების ამოსვლის ალბათობა. მაგალითად ავიღოთ „ტომ სოიერი“ და დავთვალოთ მასში თუ რა სიხშირით გვხვდება თითოეული სიმბოლო. აღმოვაჩენთ, რომ სფეისი ყველაზე ხშირი სიმბოლოა, 'e' საკმაოდ გავრცელებულია ხოლო 'q' ყველაზე იშვიათად გვხვდება. დავუშვათ სფეისი წარმოადგენს მთლიანი სიმბოლოების - 16%–ს, 'e' - 9%–ს, ხოლო 'q' - 0.04%–ს. ამ მონაცემების გამოყენებით შეგვიძლია დავაგენერიროთ შემთხვევითი ტექსტი (ისე რომ ასოს ამოსვლის ალბათობა შეესაბამებოდეს იმას თუ რამდენად ხშირად გვხვდება იგი ტექსტში). მიღებულ ტექსტს ბევრი არაფერი ექნება საერთო ტომ სოიერთან, მაგრამ სამაგიეროდ სიმბოლოები ერთი და იგივე სიხშირით იქნება ორივე ტექსტში. ქვემოთ მოცემულია ასეთი მეთოდით დაგენერირებული ტექსტის მაგალითი:

**Order 0** rla bsht eS ststofo hhfosdsdewno oe wee h .mr ae irii ela iad o r te u t mnyto

onmalysnce, ifu en c fDwn oee iteo

ეს არის მარკოვის ნულოვანი მოდელი, რომელიც გულისხმობს რომ ყოველი სიმბოლო გვხვდება ფიქსირებული ალბათობით, იმის და მიუხედავად თუ რა იყო წინა სიმბოლოები.

წარმოიდგინეთ შემთხვევით დაგენერირებული ერთი წლის ტემპერატურის პროგნოზი, საეჭვოა ვინმე ენდოს ასეთ პროგნოზს, გაცილებით დამაჯერებელი იქნება თუ ყოველი დღის ტემპერატურის პროგნოზი გაკთდება წინა დღის ტემპერატურაზე დაყრდნობით. თუ დღეს 40 გრადუსია, საეჭვოა ხვალ ტემპერატურა 15 გრადუსამდე დაეცეს. ანალოგიურად ხდება ინგლისური სიტყვებისთვისაც: თუ მოცემული გვაქვს ასო 'q', მაშინ დიდი შანსია, რომ მომდევნო ასო იყოს 'u'. გაცილებით უფრო რეალურ ტექსტს დააგენერირებთ თუ ყოველ მომდევნო სიმბოლოს არჩევისას გაითვალისწინებთ თუ რა იყო წინა სიმბოლო.

ამ პროცესისთვის უნდა აიგოს პირველი მოდელი, რომელიც დაითვლის თუ რა არის ალბათობა იმისა რომ რომელიღაცა კონკრეტული სიმბოლოს შემდეგ მოდის სხვა კონკრეტული სიმბოლო. როგორც ირკვევა 's'–ის შემდეგ უფრო დიდი შანსია ამოვიდეს 't' ვიდრე 'y', 'q'–ს შემდეგ კი თითქმის ყოველთვის 'u' გვხვდება. ახლა შეგიძლიათ კიდევ დააგენერიროთ ტომ სოიერის შემთხვევითი ტექსტი, აირჩიოთ ერთი სიმბოლო და ყოველი მომდევნო დააგენერიროთ ზემოთ მიღებული ალბათობების მიხედვით, რომელიც მიიღეთ საწყისი ტექსტის ანალიზის საფუძველზე. ქვემოთ მოცემულია ტექსტი რომელიც ამ მეთოდმა გამოიტანა:

**Order 1** "Shand tucthiney m?" le ollds mind Theybooure He, he s whit Pereg lenigabo

Jodind alllld ashanthe ainofevids tre lin--p asto oun theanthadomoere

იგივე იდეის გამოყენებით შეგვიძლია გავზარდოთ სიმბოლოების რაოდენობა. მეორე მოდელში ყოველი სიმბოლო გენერირდება იმის დამიხედვით თუ რა ორი სიმბოლო იყო მის წინ. ინგლისურ ენაში "sh" ს შემდგებ როგორც წესი ხმოვანი მოდის, ნაკლებად ხშირად 'r' და 'w' სიმბოლოები, და ძალიან იშვიათად სხვა სიმბოლო. ტომ სოიერის ტექსტის ანალიზი მეხუთე მოდელის მიხედვით გვაჩვენებს, რომ "leave" ის შემდგებ ხშირად მოდის 's' ან სფეისი, 'j' და 'q' კი არასდროს, ხოლო "Sawye" ს შემდეგ ყოველთვის მოდის 'r'. k–ური მოდელის გამოყენებისას, ყოველ შემდეგ სიმბოლოს აგენერირებთ მის წინ მდებარე k სიმბოლოს მიხედვით.

დაახლობით მეხუთე-მეშვიდე მოდელის საშუალებით მიღებული ტექსტი უკვე ბევრი რამით ემსგავსება საწყის ტექსტს. სავარაუდოდ აზრიანი ტექსტი არ იქნება მაგრამ თავისუფლად შეძლებთ იმის თქმას რომ ეს ტომ სოიერისგან მიღებული ტექსტია. უფრო მაღალ მოდელებში დაგენერირებული ტექსტი ძირითადად მართებული სიტყვებისგან შედგება და აზრსაც შეიცავს.

მაგალითები:

**Order 2** "Yess been." for gothin, Tome oso; ing, in to weliss of an'te cle -- armit.

Papper a comeasione, and smomenty, fropeck hinticer, sid, a was Tom, be

suck tied. He sis tred a youck to themen

**Order 4** en themself, Mr. Welshman, but him awoke, the balmy shore. I'll give him

that he couple overy because in the slated snufflindeed structure's kind was

rath. She said that the wound the door a fever eyes that WITH him.

**Order 6** Come -- didn't stand it better judgment; His hands and bury it again,

tramped herself! She'd never would be. He found her spite of anything the

one was a prime feature sunset, and hit upon that of the forever.

**Order 8** look-a-here -- I told you before, Joe. I've heard a pin drop. The st i l lness

was complete, how- ever, this is awful crime, beyond the village was

sufficient. He would be a good enough to get that night, Tom and Becky.

**Order 10** you understanding that they don't come around in the cave should get the

word "beauteous" was over-fondled, and that together" and decided that he

might as we used to do -- it's nobby fun. I'll learn you."

**პროგრამის მონახაზი**

თქვენმა პროგრამამ უნდა წაიკითხოს მოცემული ტექსტი, ააგოს მარკოვის k–ური მოდელი ამ ტექსტისთვის და დააგენერიროს შემთხვევითი ტექსტი.

პირველ რიგში თქვენმა პროგრამამ მომხარებელს უნდა მოთხოვოს მოცემული ტექსტური ფაილის სახელის შეყვანა, არასწორი სახელის შეყვანის შემთხვევაში პროგრამამ განმეორებით უნდა მოითხოვოს სახელის შეყვანა, სანამ მომხარებელი სწორ სახელს არ შეიყვანს. ამის შემდეგ პროგრამამ უნდა მოითხოვოს მარკოვის მოდელის ნომრის შეყვანა (1 დან 10 მდე რიცხვი).

ფაილიდან ტექსტი წაიკითეთ სიმბოლ–სიმბოლ, გამოყავით ყოველი k სიმბოლო, დააკვირდით თუ რა სიმბოლო მოდის მათ შემდეგ და დაითვალეთ ალბათობები. თქვენი ამოცანაა სიმბოლოების სიხშირის ინფორმაცია ისეთნაირად შეინახოთ, რომ შემდეგ მარიტვი ოპერაციების საშუალებით შეძლოთ ტექსტის დააგენერირება.

ტექსტის წაკითხვის შემდეგ, თქვენმა პროგრამამ უნდა გამოიტანოს შესაბამისი მოდელის მიხედვით დაგენერირებული 2000 სიმბოლოსგან შემდგარი ტექსტი. საწყისად აიღეთ მიმდევრობა რომელიც ყველაზე ხშირად გხვდებათ ტექსტში (ანუ თუ იყენებთ მეოთხე მოდელს, საწყის 4 სიმბოლოიან მიმდევრობად აიღეთ ის, რომელიც ყველაზე ხშირად გხვდებათ ტექსტში). თუ ასეთი მიმდევრობა რამოდენიმეა, აირჩიეთ რომელიმე. გამოიტანეთ საწყისი მიმდევრობა, შემდეგ გამოთვლილ ალბათობებზე დაყრდნობით აირჩიეთ მომდევნო სიმბოლო. გამოიტანეთ ახლად არჩეული სიმბოლო, განაახლეთ ის k სიმბოლო (სიმბოლოები რომლთა მიხედვითაც ხდება მომდევნო სიმბოლოს არჩევა) და გაიმოერეთ იგივე სანამ არ დააგენერირებთ 2000 სიმბოლოს.

მაგალითად, განვიხილოთ მარკოვის მეორე მოდელი, მოცემული ტექსტისთვის:

As Gregor Samsa awoke one morning from uneasy dreams he found himself transformed in his bed into a gigantic insect.

ქვემოთ მოცემულია, თუ როგორ შეიძლება შეირჩეს პირველი რამოდენიმე სიმბოლო:

1. ყველაზე ხშირად ოთხჯერ გვხვდება მიმდევრობა "in", ასე რომ მას ვირჩევთ საწყის მიმდევრობად.
2. შემდეგ სიმბოლოს ვირჩევთ იმის მიხედვით თუ რა სიმბოლოები გვხვდება ტექსტში "in" მიმდევორბის შემდეგ. "in" ტექსტში გვხვდება ოთხჯერ, პირველ შემთხვევაში მის შემდეგ მოდის 'g', მეორე შემთხვევაში სფეისი, მესამე და მეოთხე შმეთხვევებში 't' და 's'. გამოდის რომ თითოეულის არჩევის ალბათობა არის 1/4. დავუშვათ ამ შემთხვევაშ ავირჩიეთ სფეისი.
3. აქტიური მიმდევრობა გახდა "n ". მოცემული ტექსტი შეიცავს ერთ ასეთ მიმდევრობას, რომლის შემდეგაც მოდის 'h', ასე რომ შემდეგ სიმბოლო არის 'h'.
4. ახალი აქტიური მიმდევრობა არის " h". ტექსტი შეიცავს სამ ასეთ მიმდევრობას, ერთის შემდეგ მოდის 'e' და დანარჩენი ორის შემდეგ 'i'. გამოდის რომ 'e' არჩევის ალბათობა არის 1/3 ხოლო 'i' ს არჩევის 2/3. დავუშვათ რომ ავირჩიეთ 'i'.
5. აქტიური მიმდევრობა გახდა "hi". ტექსტში ორი ასეთი მიმდევრობა გვხვდება, ერთის შემდეგ მოდის 'm' მეორის შემდეგ კი 's'. მომდევნო სიმბოლოსთვის თანაბარი შანსია (1/2) 'm' ის და 's' ამოსვლის.
6. და ასე შემდეგ.

თუ თქვენ პროგრამას ექნება სიტუაცია როცა არ არის ასარჩევი სიმბოლო (ეს ხდება მხოლოდ მაშინ, როცა აქტიური მიმდევრობა ტექსტში გვხვდება ერთადერთხელ, ტექსტის ბოლოს), თქვენ პროგრამას შეუძლია ნაადრევად შეწყვიტოს ტექსტის გამოტანა.

**სასარგებლო მინიშნებები**

1. Map დაVector არის ზუსტად ის რაც გჭირდებათ ინფორმაციის შესანახად. მეფის გასაღები იქნება k სიგრძის მიმდევრობა (ანუ თუ განვიხილავთ მარკოვის მეორე მოდელს, თითოეული გასაღები იქნება ორ სიმბოლოიანი მიმდევრობა, რომელიც გვხვდება მოცემულ ტექსტში). მნიშვნელობა კი იქნება ყველა იმ სიმბოლოებისგან შემდგარი ვექტორი, რომლებიც სიდის შემდეგ გვხვდებიან ტექსტში. ვექტორში შეიძლება იყოს დუბლირებული სიმბოლოები და სავარაუდოდ ასეც იქნება. ცხადია რაც უფრო მეტჯერ გვხვდება სიმბოლო ვექტორში მით უფრო დიდია მისი ამოსვლის ალბათობა. ვექტორში დუბლიკატების შენახვა საშუალებას მოგვცემს მარტივად შევარჩიოთ რანდომ სიმბოლო(ზოგადად ახალი სიმბოლოს არჩევის ალბათობები ერთიდაიგივე არ არის და დამოკიდებულია სიმბოლოების სიხშირეზე). უფრო ეფექტური იქნება თუ ყოველი სიმბოლოსთვის შევინახავთ რაოდენობას თუ რამდენჯერ გვხვდება იგი k სიგრძის მიმდევრობაში. მაგრამ ამ შემთხვევაში შედარებით მოუხერხებელია პროგრამის რეალიზაციის დაწერა.
2. მეფის შედგენის შემდეგ ადვილად შეძლებთ იმ მიმდევრობის პოვნას, რომელიც ყველაზე ხშირად გვხვდება ტექსტში.
3. ფაილის სიმბოლ-სიმბოლ წასაკითხად გამოიყენეთ ifstream სტრიმის get ფუნქცია.

**დავალების შესრულების გეგმა**

1. გაუშვით დემო პროგრამა და ნახეთ თუ როგორ მუშაობს იგი.
2. დავალების შესრულებისთვის დაგჭირდებათ Vector და Map კლასების გამოყენება, ასე რომ პირველ რიგში გაეცანით მათ მეთოდებს.
3. მოიფიქრეთ მონაცემთა სტრუქტურა, სადაც შეინახავთ ტექსტის ანალიზის შედეგებს. აუცილებელია კარგად გესმოდეთ, როგორ უნდა აიგოს ერთმანეთში ჩადგმული string/vector/map ობიექტებისგან შედგენილი სტრუქტურა.
4. დაწერეთ ტექსტის წაკითხვის და ანალიზის რეალიზაცია. დარწმუნდით რომ სწორად მუშაობს ტექსტის ანალიზი და მონაცემების შენახვა და მხოლოდ ამის შემდეგ გადადით შემდეგ პუნქტზე.
5. დავალების დიდი ნაწილი, ტექსტის წაკითხვა და ანალიზი უკვე შესრულებული გაქვთ, ასე რომ შეგიძლიათ გადახვიდეთ შემთხვევითი ტექსტის გენერატორის დაწერაზე.

შეთხვევითი ტექსტის გენერატორი შეგიძლიათ გამოიყენოთ ნებისმიერი ტიპის ტექსტისთვის. საწყის ტექსტად შეგიძლიათ ქართული ტექსტიც გამოიყენოთ, შეამოწმეთ რას დააგენერირებს თქვენი პროგრამა სხვადასხვა ტექსტებისთვის.