***გამოცდის ფორმატი***

**\*მონიშნეთ გამოცდის ფორმატი (მიუთითეთ √)**

|  |  |
| --- | --- |
| დახურული წიგნი |  |
| ღია წიგნი | **√** |
| **\*ღია წიგნის შემთხვევაში მონიშნეთ გამოცდაზე ნებადართული ელემენტები (მიუთითეთ √)** | |
| სალექციო მასალები (პრეზენტაცია და სხვა) |  |
| ელექტრონული წიგნები | **√** |
| წიგნები |  |
| კონსპექტები |  |
| ლექსიკონი |  |
| კალკულატორი |  |
| ლეპტოპი/პლანშეტი |  |

**\* გამოცდის ჩატარების წესი იხილეთ ,,დესკტოპზე“ საქაღალდეში Exam materials**

***საგამოცდო საკითხების ფორმა***

***ვარიანტი # 1***

|  |  |  |  |
| --- | --- | --- | --- |
| ***სკოლა*/*საგანმანათლებლო პროგრამა*** | მათემატიკა და კომპიუტერული მეცნიერება | ***სტუდენტის მიერ მიღებული ქულა*** |  |
| ***საგანი*** | პროგრამირების აბსტრაქციები | | |
| ***ლექტორი*** | გ. ბოჭორიშვილი | | |
| ***კურსი*** | I | | |
| ***ჯგუფი*** |  | | |
| ***გამოცდის ფორმა*** | ღია წიგნი | | |
| ***გამოცდის***  ***ხანგრძლივობა*** | 3 საათი | | |
| ***მაქსიმალური ქულა*** | 180 | | |
| ***სტუდენტის სახელი და გვარი:*** | | | |

სახელი: ქულა:

ფინალური გამოცდა (180 ქულა)

პროგრამირების აბსტრაქციები

10 ივლისი, 15:00-18:00

**შენიშვნა: გამოცდიდან სტუდენტის მოხსნა ნებისმიერი მიზეზით ავტომატურად ნიშნავს, საგანში ფეილს.**

**ამოცანა 1. (20 ქულა)**

რას დაბეჭდავს ქვემოთ მოცემული პროგრამები. ჩათვალეთ რომ ყველა საჭირო ინკლუდი გაკეთებულია. თუ ფიქრობთ, რომ პროგრამაში არის შეცდომა ახსენით შეცდომის მიზეზი.

ა)

void a(int p, int &k) {

k = p + 1;

p = k + p;

}

void b(int p, int \*k) {

p = 'c' - 'b';

\*k = \*k + p;

}

int main() {

int k = 9;

int p = 3;

a(p, k);

b(p, &k);

cout << k << endl;

cout << p << endl;

return 0;

}  
**პასუხი:**

ბ)

void a(char &c) {

cout << c << endl;

c = 'a';

cout << c << endl;

}

int main() {

a('e');

return 0;

}  
**პასუხი:**

გ)

int rec1(int p) {

if (p < 0)

return 1;

if (p == 0)

return 2;

return rec1(p-1) + rec1(p-2);

}

int rec2(int &p) {

if (p < 0)

return 1;

if (p == 0)

return 2;

return rec1(p-2) + rec1(p-1);

}

int main() {

int p = 3;

cout << rec1(p) << endl;

cout << rec2(p) << endl;

return 0;

}  
**პასუხი:**

დ)  
int cmp(int a, int b) {

return (a/b) \* (b/a);

}

int main() {

Set<int> s(cmp);

if (s.contains(10)) {

cout << "10" << endl;

}

s.add(1);

s.add(2);

s.add(2);

s.add(3);

s.add(4);

s.add(5);

cout << "size: " << s.size() << endl;

if (s.contains(1)) {

cout << "1" << endl;

}

if (s.contains(2)) {

cout << "2" << endl;

}

if (s.contains(7)) {

cout << "7" << endl;

}

if (s.contains(8)) {

cout << "8" << endl;

}

return 0;

}

**პასუხი:**

**ამოცანა 2.(35 ქულა) ხის ბეჭდვა**

მოცემული გვაქვთ ორობითი ძებნის ხე (BST). თქვენი მიზანია, დაწეროთ მეთოდი, რომელსაც გადაეცემა ხის root ელემენტი და ბეჭდავს ხის ელემენტებს შემდეგი თანმიმდევრობით: მინიმალური, მაქსიმალური, დარჩენილებს შორის მინიმალური, დარჩენილებს შორის მაქსიმალური...

მაგალითად თუ ხეში ჩაგდებული იქნება 1, 5, 9, 13, 16, 19 თქვენი გამოტნილი შედეგი უნდა იყოს: 1, 19, 5, 16, 9, 13. გაითვალისწინეთ, რომ არანაირი დამატებითი სტრუქტურის გამოყენების უფლება არ გაქვთ. არსებული ხის შეცვლა შეგიძლიათ.

Struct NodeT{

Int val;

NodeT \* left;

NodeT \* right;

}

Void printTree(NodeT \* head);

**ამოცანა 3.(25 ქულა) სელექშენ სორტი**

თქვენი ამოცანაა დაწეროთ ბმული სიის სორტირება(val ის მნიშვნელობის მიხედვით) სელექშენ სორტის ალგორითმით. ქვემოთ მოცემული გაქთ სელექშენ სორტირების ფსევდო კოდი:

int i,j;

int iMin;

for (j = 0; j < n-1; j++) {

iMin = j;

for ( i = j+1; i < n; i++) {

if (a[i] < a[iMin]) {

iMin = i;

}

}

if(iMin != j) {

swap(a[j], a[iMin]);

}

}

გაითვალისწინეთ, რომ არ გაქთ new ოპერატორის გამოყენების უფლება, ასევე არ შეიძლება ახალი სტრუქტურების შექმნა როგორიცაა Vector, Set, Map, Stack, Queue და თვითონ Entry სტრუქტურაც. სია უნდა დაასორტიროთ მხოლოდ არსებული ელემენტების გადამისამართების საშუალებით.

struct Entry {  
 int val;  
 Entry \*next, \*prev;  
}

void selectionSortLinkedList(Entry \* &head);

**ამოცანა 4. (30 ქულა) კუბის დეკომპოზიცია**

მოცემული მთელი რიცხვის კუბს ეწოდება იდეალური თუკი შეიძლება მისი წარმოდგენა სხვა **განსხვავებული** კუბების ჯამის სახით. მაგალითად:

6^3 = 5^3 + 4^3 + 3^3

თქვენი მიზანია გაიგოთ არის თუ არა მოცემული რიცხვის კუბი იდეალური. დაწერეთ ფუნქცია რომელსაც გადაეცემა მთელი დადებითი რიცხვი n და ცარიელი ინტების სიმრავლე. თუკი n^3 წარმოდგება განსხვავებული მთელი რიცხვების კუბების სახით, მაშინ ფუნქციამ უნდა დააბრუნოს მნიშვნელობა true, ხოლო სიმრავლეში უნდა იყოს ის მთელი რიცხვები რომელთა კუბების ჯამიც n^3-ის ტოლია. თუკი წარმოდგენა შეუძლებელია მაშინ ფუნქციამ უნდა დააბრუნოს false ხოლო სიმრავლე უნდა იყოს ცარიელი. ყურადღება მიაქციეთ იმ ფაქტს, რომ თუკი შესაძლებელია ერთიდაიგივე რიცხვის განსხვავებული კუბების ჯამის სახით წარმოდგენა, მაშინ პასუხად უნდა აირჩიოთ ის სიმრავლე რომელიც შეიცავს მინიმალური რაოდენობის ელემენტს.

bool idealCube(int n, Set<int>& cubes){

**ამოცანა 5.** ხე თუ ბუჩქი? (30 ქულა)

struct Node {

string town;

Node \* left, \*right;

};

Node-ის საშუალებით მოცემული გაქვთ ხის მსგავსი სტრუქტურა, რომელიც შეიძლება სულაც არ იყოს ხე, რადგან შეიძლება ჰქონდეს ციკლები და ა.შ.

თქვენი ამოცანაა დაწეროთ isTreeRoot ფუნქცია, რომელსაც გადაეცემა სტრუქტურის სათავე და რომელიც ამოწმებს ნამდვილად არის თუ არა ეს სტრუქტურა ხე.

შეგახსენებთ: მსგავსი სტრუქტურა არის ხე მაშინ და მხოლოდ მაშინ თუ:

1. ის არ შეიცავს ციკლებს.
2. სათავიდან ყველა წვერომდე არსებობს ერთადერთი გზა.

bool isTreeRoot(Node \* head);

**ამოცანა 6. (40 ქულა) საწყალი პაციენტები**

MultQueue კლასი აღწერს რამოდენიმე პრიორიტეტული რიგის მუშაობის პროცესს. სიმარტივისთვის შეგიძლიათ წარმოიდგინოთ საოპერაციოს მუშაობის პრინციპი. საავადმყოფოში არიან ექიმები რომელბსაც ჰყავთ პაციენტები. ეს პაციენტები შედიან ექიმთან გარკვეული პრიორიტეტის მიხედვით. როდესაც ახალი პაციენტი მოდის მიმღებში, ადგენენ მის პრიორიტეტს და შემდეგ სვამენ პაციენტს რომელიმე ექიმის სიაში, ისე რომ რიგში მის წინ არსებული პაციენტების რაოდენობა მინიმალურია და მის წინ არსებული ყველა პაციენტი მასზე უფრო პრიორიტეტულია.

კლასს აქვს შემდეგი ფუნქციები:

**void addQueue();** - ამატებს ახალ პრიორიტეტულ რიგს.

**string dequeue(int i);** - i ინდექსის მქონე რიგიდან ყველაზე პრიორიტეტული ელემენტის ამოღება.

**void enqueue(string s, int p);** - ამატებს კლასში p პრიორიტეტის მქონე ელემენტს, ელემენტი უნდა ჩაემატოს არსებული რიგებიდან ისეთში სადაც უკეთესი პოზიცია ექნება. რაც უფრო ნაკლები dequeue ჭირდება ელემენტის ამოღებას, მით უკეთესია პოზიცია.

**int size();** - აბრუნებს ამ მომენტისთვის არსებული ელემენტების რაოდენობას.  
**void removeQueue(int i);** - შლის რიგს და რიგში არსებულ ელემენტებს ანაწილებს დანარჩენ რიგებში.

თქვენი მიზანია მოიფიქროთ MultQueue კლასის მონაცემთა სტრუქტურა და დაწეროთ ფუნქციების რეალიზაცია. თავდაპირველად MultQueue შედგება 2 რიგისგან.