**Sinov turlarini o'rganish. Sinovni rejalashtirish**

Agar siz uni bir marta ishga tushirish va uni unutish uchun dastur yozgan bo'lsangiz, bundan keyin hech narsa qilishingiz shart emas. Bu yerda sizning dasturingiz ba'zi hollarda to'g'ri ishlamasligi juda muhim emas: agar birinchi boshlanganda u siz kutgandek ishlasa va agar siz uni yanada ko'proq ishlatishni va ishlatishni rejalashtirmasangiz, unda hamma narsa tugadi.

Dasturiy ta'minotni sinovdan o'tkazish - bu dasturiy ta'minot ishlab chiquvchi kutganidek ishlashini aniqlash jarayonidir.

O'zingizning kodingizni sinab ko'rishning ba'zi bir amaliy usullari haqida gapirishdan oldin, keling, nima uchun murakkab sinovlar qiyin kechishi mumkinligini quyidagi dasturni ko'rib chiqamiz:

#include <iostream>

#include <string>

using namespace std;

void compare(int a, int b) {

if (a > b) cout << a << " katta " << b << "dan "<< '\n'; // №1 hodisa

else if (a < b) cout << a << " kam " << b << "dan "<< '\n'; // №2 hodisa

else cout << a << " teng " << b << "ga "<< '\n'; }// №3 hodisa

int main(){

cout << "Raqamni kiriting: ";

int a; cin >> a;

cout << "Boshqa raqamni kiriting: ";

int b; cin >> b;

compare(a, b);}

4 baytli int turi va uning qiymatlari oralig'ini hisobga olgan holda , biz barcha mumkin bo'lgan qiymatlarni sinab ko'rish uchun ushbu dasturni 18 446 744 073 709 551 616 (~ 18 kvintillion) marta bajarishimiz kerak. Bu bema'ni ekanligi aniq.

Biz har safar foydalanuvchini kiritishni so'raganimizda yoki dasturda shartli tarmoqlanishdan foydalanganimizda, biz dasturimizni amalga oshirishning mumkin bo'lgan usullarini ko'paytiramiz. Oddiy dasturlardan tashqari barcha dasturlar uchun kiritilgan ma'lumotlarning har bir kombinatsiyasini va hatto qo'lda sinab ko'rish - bu qandaydir mantiqqa to'g'ri kelmaydi, deb o'ylaysizmi?

Norasmiy testlarni o'tkazish

Ko'pgina dasturchilar o'z dasturlarini yozishda norasmiy testlarni o'tkazadilar . Kod qismini (funktsiya, sinf yoki boshqa biron bir "kod bo'lagi") yozgandan so'ng, ishlab chiquvchi yangi qo'shilgan qismni sinab ko'rish uchun ba'zi bir kodlarni yozadi va agar test muvaffaqiyatli bo'lsa, ishlab chiquvchi ushbu test uchun kodni o'chirib tashlaydi. Masalan, quyidagi isLowerVowel () funktsiyasi uchun tekshirish uchun quyidagi kodni yozishimiz mumkin:

#include <iostream>

using namespace std;

bool isLowerVowel(char c) {

switch (c) {

case 'a': case 'e': case 'i':

case 'o': case 'u':

return true; default:

return false; } }

int main() {

cout << isLowerVowel('a'); // vaqtinchalik sinov kodi, natijada 1 bo'lishi kerak

cout << isLowerVowel('q'); // vaqtinchalik sinov kodi, natijada 0 bo'lishi kerak

}

Agar siz dasturni ishga tushirganingizda 1va olsangiz 0, unda hamma narsa yaxshi. Siz o'zingizning funktsiyangiz ishlayotganini bilasiz, shuning uchun vaqtinchalik sinov kodini olib tashlashingiz va dasturlash jarayonini davom ettirishingiz mumkin.

Maslahat №1: Dasturingizni parcha-parcha qilib yozing: kichik, aniq belgilangan birliklarda (funktsiyalarda)

Masalan, avtomobil ishlab chiqaradigan avtomobil ishlab chiqaruvchisini olaylik. Sizningcha, u qaysi ishlarni bajaradi?

O'rnatishdan oldin har bir transport vositasini alohida-alohida quradi (yoki sotib oladi) va sinovdan o'tkazadi. Komponent muvaffaqiyatli sinovdan o'tkazilgandan so'ng, avtoulov ishlab chiqaruvchisi uni vositaga birlashtiradi va integratsiya muvaffaqiyatli bo'lishini ta'minlash uchun qayta sinovdan o'tkazadi. Oxir-oqibat, taqdimotdan oldin butun transport vositasining umumiy ishlash sinovi o'tkaziladi.

Barcha tarkibiy qismlardan avtoulovni hech qanday oldindan tekshiruvsiz (bitta harakat bilan) quradi. Keyin, oxirida allaqachon yig'ilgan avtomobilning ishlashining birinchi va yakuniy sinovlari o'tkaziladi.

Birinchi variantni to'g'riroq deb o'ylamaysizmi? Va shunga qaramay, ko'pchilik yangi boshlagan dasturchilar o'zlarining kodlarini ikkinchi variantga muvofiq yozadilar!

Ikkinchi holatda, agar mashinaning biron bir qismi to'g'ri ishlamasa, unda mexanik nima sodir bo'lganligini aniqlash uchun butun mashinani tashxislashi kerak bo'ladi - muammo hamma joyda bo'lishi mumkin. Masalan, nosoz sham, akkumulyator, yonilg'i pompasi yoki boshqa biron bir narsa tufayli mashina ishlamay qolishi mumkin. Bu muammoning ildizini aniqlash uchun juda ko'p vaqtni behuda sarflaydi. Va agar muammo aniqlansa, oqibatlari halokatli bo'lishi mumkin: avtomobilning bir qismidagi o'zgarishlar "kapalak effektiga" olib kelishi mumkin - avtomobilning boshqa qismlarida katta o'zgarishlar. Masalan, juda kichik bo'lgan yonilg'i pompasi dvigatelni o'zgartirishi mumkin, bu esa avtomobilning ramkasini qayta tashkil etishga olib keladi. Oxir oqibat, siz mashinaning katta qismini qayta tiklashingiz kerak bo'ladi,

Birinchi holda, avtomobil ishlab chiqaruvchisi barcha qismlarni mavjud bo'lganda tekshiradi. Agar tarkibiy qismlardan birortasi nuqsonli bo'lib chiqsa, unda mexanika darhol muammo nima ekanligini va uni qanday hal qilishni tushunadi. Muvaffaqiyatli sinovdan o'tmaguncha, hech narsa mashinaga qo'shilmaydi. Ular butun avtoulovni yig'ishganida, uning ishlashiga o'rtacha ishonch hosil bo'ladi - axir barcha qismlar muvaffaqiyatli sinovdan o'tkazildi. Barcha qismlarni ulashda xatolik yuz berishi ehtimoli hali ham mavjud, ammo ikkinchi variant bilan taqqoslaganda, bu siz uchun jiddiy tashvishlanmaslik kerak bo'lgan juda kichik ehtimollik.

Yuqoridagi o'xshashlik dasturchilar uchun ham amal qiladi, ammo ba'zi bir sabablarga ko'ra yangi boshlanuvchilar buni sezmaydilar. Kichik funktsiyalarni yozib, so'ngra ularni darhol tuzish va sinovdan o'tkazish yaxshiroqdir. Shunday qilib, agar siz xato qilsangiz, bu siz yozgan / o'zgartirgan oz miqdordagi kodda ekanligini bilib olasiz. Va bu, o'z navbatida, xatolarni qidirish maydoni kichikligini va disk raskadrovka uchun juda kam vaqt sarflanishini anglatadi.

Maslahat №2: kodni 100% qamrab olishga intiling

Muddatli "kodi qamrov" sinash paytida ishlatilgan dastur manba kod miqdorini anglatadi. Kodni qamrab olish ko'rsatkichlari juda ko'p, ammo ulardan ba'zilari haqida aytib o'tish joiz.

Vaqtni qoplash - bu sizning kodingizdagi testlar paytida ishlatilgan fikrlarning foizidir. Masalan:

int boo(int a, int b) {

bool z = b;

if (a > b) {

z = a; }

return z; }

Murojaat boo(1, 0)sizga ushbu funktsiya vaqtni qoplash to'liq qamrab oladi, chunki har bir kod satri bajariladi.

IsLowerVowel () funktsiyasi holatida:

bool isLowerVowel(char c) {

switch (c) { // bayonot №1

case 'a': case 'e': case 'i': case 'o': case 'u':

return true; // bayonot №2

default:

return false; } } // bayonot №3

Barcha bayonotlarni tekshirish uchun ikkita murojaat kerak bo'ladi, chunki biz bitta funktsiya chaqirig'ida # 2 va # 3 so'zlarning ishlashini aniqlay olmaymiz.

Maslahat №3: 100 foiz filialni qamrab olishga intiling

Muddatli "filiali qamrov" alohida-alohida har bir holatda (ijobiy va salbiy) bajarildi filiallari foizini bildiradi. Kommutator bayonoti ko'plab filiallarga ega bo'lishi mumkin. If ifodasi ikkita shoxdan iborat: haqiqiy holat va yolg'on holat (agar boshqa bayonot bo'lmasa ham). Masalan:

int boo(int a, int b) {

bool z = b;

if (a > b) {

z = a; }

return z; }

Oldingi qo'ng'iroq boo(1, 0)bizni 100% bayonot bilan qamrab oldi va haqiqiy dallanishni ta'minladi. Ammo bu filiallarning 50 foiz qamrab olinishi. Biz yana bitta qo'ng'iroqni talab qilamiz - boo(0, 1)tarmoqlanishning noto'g'ri ekanligini tekshirish uchun.

IsLowerVowel () funktsiyasi filiallarning 100% yopilganligiga ishonch hosil qilish uchun ikkita qo'ng'iroqni talab qiladi (masalan, isLowerVowel('a')va isLowerVowel('q')) (tugmachadagi barcha harflarni sinash shart emas, agar ular ishga tushirilsa, boshqalari ham ishlaydi):

#include <iostream>

using namespace std;

bool isLowerVowel(char c) {

switch (c) {

case 'a': case 'e': case 'i':

case 'o': case 'u':

return true;

default:

return false; } }

int main() {

cout << isLowerVowel('a'); // vaqtinchalik sinov kodi, natijada 1 bo'lishi kerak

cout << isLowerVowel('q'); } // vaqtinchalik sinov kodi, natijada 0 bo'lishi kerak

Ushbu qo'llanmaning birinchi misolidan taqqoslash funktsiyasini qayta ko'rib chiqamiz:

void compare(int a, int b) {

if (a > b)

std::cout << a << " katta " << b <<” dan ”<< '\n'; // случай №1

else if (a < b)

std::cout << a << " kichik " << b << ” dan ”<<'\n'; // случай №2

else

std::cout << a << " teng " << b <<” ga ”<< '\n'; // случай №3

}

Filialni 100% qamrab olish uchun 3 ta funktsiya murojaati kerak:

compare(1,0) birinchi if ifodasi uchun rostligini tekshiradi.

compare(0, 1) birinchi if ifodasi uchun false, ikkinchisi uchun true (agar boshqa bo'lsa) uchun tekshiriladi.

compare(0, 0) if ifasi uchun false ni tekshiradi va else ifodasini bajaradi.

Shunday qilib, biz ushbu funktsiyani faqat 3 ta funktsiya chaqiruvi bilan sinab ko'rishimiz mumkin (18 kvintillion marta emas).

Qoida: Dasturingizdagi har bir tarmoqlangan ishni sinab ko'ring.

Maslahat №4: tsiklni 100% qamrab olishga intiling

Loop qamrovi (norasmiy ravishda "test 0, 1, 2" deb nomlanadi ) sizning kodingizda tsikl bo'lsa, uni ishlashiga ishonch hosil qilish uchun uni 0, 1 va 2 marta ishlatishingiz kerakligini aytadi. Agar u ikkinchi takrorlashda to'g'ri ishlayotgan bo'lsa, unda barcha keyingi takrorlashlar uchun to'g'ri ishlashi kerak (3, 4, 10, 100 va boshqalar). Masalan:

#include <iostream>

int spam(int timesToPrint)

{

for (int count=0; count < timesToPrint; ++count)

std::cout << "Spam!!!";

}

Funktsiya ichidagi tsiklni sinab ko'rish uchun uni 3 marta chaqirishimiz kerak:

spam(0)nol takrorlanish holatini tekshirish uchun.

spam(1)# 1 spam(2)takrorlashni va # 2 takrorlashni sinash uchun.

Agar u spam(2)ishlasa, u spam(n)ishlaydi (qaerda n > 2).

Qoida: Turli xil takrorlashlar bilan to'g'ri ishlashini tekshirish uchun "test 0, 1, 2" dan foydalaning.

**Sinovlarni saqlash**

Sinovlarni yozish va keyin ularni yo'q qilish tez va vaqtinchalik sinov uchun juda yaxshi imkoniyatdir, chunki kelajakda qayta ishlatishni yoki o'zgartirishni niyat qilgan kod uchun ushbu testlarni saqlab qolish mantiqan to'g'ri keladi. Masalan, vaqtinchalik testni o'chirish o'rniga uni test () funktsiyasiga o'tkazishingiz mumkin:

#include <iostream>

bool isLowerVowel(char c)

{ switch (c) {

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

return true;

default:

return false; } }

// Ushbu funktsiya hozirda biron bir joyda chaqirilmaydi, lekin agar siz qayta sinov qilmoqchi bo'lsangiz.

void test()

{

std::cout << isLowerVowel('a'); // vaqtinchalik sinov kodi, natijada 1 bo'lishi kerak

std::cout << isLowerVowel('q'); // vaqtinchalik sinov kodi, natijada 0 bo'lishi kerak

}

int main() {

return 0;

}

Sinovlarni avtomatlashtirish

Yuqorida aytib o'tilgan test funktsiyasi bilan bog'liq muammolardan biri shundaki, siz test natijalarini qo'lda tekshirishingiz kerak bo'ladi. Va siz yaxshiroq qilishingiz mumkin - muvaffaqiyatli test bilan olinishi kerak bo'lgan testga to'g'ri kutilgan natijalarni qo'shing:

#include <iostream>

bool isLowerVowel(char c) {

switch (c) {

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

return true;

default:

return false; } }

// Muvaffaqiyatsiz bo'lgan test raqamini yoki barcha testlar muvaffaqiyatli o'tgan bo'lsa 0 qiymatini qaytaradi

int test() {

if (isLowerVowel('a') != true) return 1;

if (isLowerVowel('q') != false) return 2;

return 0; }

int main() {

return 0; }

Endi siz istalgan vaqtda test () ga murojaat qilishingiz mumkin va funksiya siz uchun hamma narsani qiladi.

**Topshiriq**

**Sinov mavzusi bo’yicha mustaqil 4 tadan misol ishlash.**