Zapiski s priprav

Reševanje nalog na papir

ACM priprave na tekmovanja

Različica z dne 17. januar 2025

Kazalo

1 Tekmovanje RTK

Na šolskem nivoju tekmovanja RTK ter v nekaterih skupinah tekmovanja na državnem nivoju se programerske naloge rešuje „na papir.“ To ne pomeni, da moraš rešitve dejansko z roko napisati na list papirja, temveč le, da oddaje ne bodo avtomatsko ocenjene, temveč jih bo pregledal eden od popravljavcev, rezultat pa dobiš šele po koncu tekmovanja. Pišeš lahko bodisi na računalnik, bodisi na list papirja.

Ker tvojega programa ne bo nihče prevedel ali pognal, so sprejemljive tudi rešitve, ki niso popolnoma sintaktično pravilne — za napake kot manjkajoče podpičje na koncu vrstice, napačen formatnik v `printf`, tipkarski škrati ipd. se praviloma odšteje le nekaj točk, kljub temu da tak program na običajnem tekmovanju nebi dosegel nobene točke. Med drugim to tudi pomeni, da lahko oddajaš delne rešitve. Če nalogo na primer razdeliš na dva manjša problema, enega od katerih znaš rešiti, drugega pa ne, lahko zapišeš rešitev prvega podproblema in opišeš, kako bi jo lahko kombiniral z rešitvijo drugega problema. Za tako oddajo seveda ne dobiš vseh točk, če pa je rešen del problema dovolj pomemben, pa ti popravljavci lahko dodelijo delne točke.

Ker programa ne bo pregledal računalnik, pač pa človek, moraš svojo rešitev napisati tako, da bo bralcu jasno, kaj tvoja rešitev počne. To med drugim pomeni, da uporabljaš deskriptivna imena spremenljivk, da v komentarjih programa razložiš, kakšno idejo tvoja rešitev uporablja, ter da na kratko utemeljiš, kaj naredi posamezen blok kode. Koda mora biti tudi lepo poravnana, da bo bralec lahko že s pogledom videl, kako je program strukturiran.

Včasih navodila naloge dopuščajo pisanje rešitev v psevdokodi. To pomeni, da lahko rešitev zapišeš v besedah, a jo strukturiraš kot da bi bila koda — torej tako, da lahko vsakdo, ki je vsaj malo vešč v programiranju, tvoj zapis enostavno prevede v svoj najljubši programski jezik. Ko pišeš v psevdokodi, uporabljaš standardne strukturne elemente programov (`if` stavke, `for` zanke itd.), vendar jih zapišeš v slovenščini, ne kot kodo.

2 Primeri

Spodaj je nanizanih nekaj primerov rešitev za naloge iz šolskega tekmovanja MladiRTK 2024, ki so dostopne na tej povezavi. Vse naloge so vredne 25 točk.

2.1 Pribor

Kot prvi primer si poglejmo program spodaj. Iz opisa je razvidno, da je tekmovalec prebral nalogo, in da ve, kateri funkciji se v C++ uporabljata za branje in pisanje iz standardnega vhoda in izhoda, vendar je zapisana rešitev vredna 0 točk. Tekmovalec ni nakazal, kako bi program dejansko napisal, temveč je le v svojih besedah zapisal besedilo naloge.

```
1 #include <stdio.h>
2
```

```c
int main() {
    int n;
    scanf("%d", &n);  // s scanfom preberemo število n
    // potem bom nkrat prebral število in ga shranil v int x.
    // Vsakič ko preberem večje število od velikosti trenutnega kupa
    // bom začel nov kup, in nanj zlagal žlice tako dolgo, dokler ne
    // dobim x, ki je večji od najmanjše žlice na kupu, takrat pa spet
    // začnem nov kup.
    // To ponavljam, dokler ne pridem do n, ko končam program in s
    // printf izpišem rešitev
    return 0;
}
```

Primerjajmo to z rešitvijo, ki je vredna vseh 25 točk. Ne samo, da spodnja koda pravilno reši problem, to dejstvo je tekmovalec tudi utemeljil. Koda je berljiva, poleg tega pa je na vsakem koraku utemeljeno, kakšno vlogo ima vsak del kode.

```c
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);

    // Ideja: hranimo si velikost prejšnje žlice in trenutno velikost
    // kupa. Ko preberemo manjšo žlico od trenutne, bomo izpisali
    // velikost kupa in začeli nov kup s to žlico.

    int prejsnja;  // velikost prejšnje žlice
    int velikost = 1;  // število žlic v trenutnem kupu
    // Prvo žlico obravnavamo posebej, in jo bomo vedno dali na
    // začetni kup, zato lahko začnemo z velikostjo 1, in beremo
    // direktno v prejsnja
    scanf("%d", &prejsnja);

    // V zanki beremo velikosti. Ker smo eno velikost že prebrali,
    // iteriramo le do n-1
    for (int i = 0; i < n-1; i++) {
        int trenutna;
        scanf("%d", &trenutna);
        if (trenutna > prejsnja) {
            // začnemo nov kup, na katerega že položimo trenutno
            // žlico
```

```
            printf("%d\n", velikost);
            velikost = 1;
        } else {
            // žlico lahko damo na isti kup
            velikost++;
        }
        prejsnja = trenutna;
    }

    // zadnjega kupa še nismo izpisali, ker nismo nanj nikoli
    // poskusili položiti prevelike žlice
    printf("%d\n", velikost);
    return 0;
}
```

Spodnja rešitev namesto programa poda opis rešitve. Naloga je zahtevala kodo ali psevdokodo, zato za opisno rešitev odbijemo nekaj točk, a je iz spodnjega opisa jasno, da tekmovalec razume, kako bi kodo rešitve dejansko zapisal, zato še vedno prejme delne točke. Žal pa so tudi v opisu napake, za katere dodatno odbijemo nekaj točk. Tekmovalec je števila na vhodu shranil v spomin, za kar odbijemo 2 točki, kakor za to nalogo priporočajo nasveti za popravljanje, poleg tega pa je tekmovalec pozabil posebej obravnavati prvo prebrano število, za kar odbijemo 4 točke. Opisan postopek tudi ne omeni, da moramo na koncu še izpisati višino zadnjega stolpa, izpis namreč zahteva le ob obravnavi prevelike žlice. Ta rešitev je tako vredna 10 točk.

```
Prvo preberemo število n, in nato z zanko preberemo še preostala
števila na vhodu, ki si jih shranimo v seznam.
Ustvarimo še eno pomožno spremenljivko, ki predstavlja velikost
trenutnega stolpa žlic. Ker na začetku nimamo žlic, to spremenljivko
nastavimo na 0.

Potem v zanki pregledujemo seznam števil na vhodu. Vsakič, ko zanka
pride v glavni del, preverimo, če je trenutna žlica manjša ali enaka
prejšnji žlici. Če je, potem žlico damo na trenutni kup, torej
povečamo pomožno spremenljivko za 1, če pa ni, potem pa izpišemo
velikost trenutnega stolpa. Potem pomožno spremenljivko spet nastavimo
na 0 in na stolp damo to žlico, torej povečamo spremenljivko za 1.
```

Poglejmo si še eno rešitev v psevdokodi. Če bi spodnje besedilo prepisali v C++, bi bila rešitev pravilna, razen tega, da bi tekmovalec za primerjavo moral uporabiti <= namesto <, saj v navodilih piše, da lahko zložimo dve enako veliki žlici eno nad drugo. V

navodilih za popravljanje je navedeno, da za tako napako odbijemo 5 točk. Načeloma bi bila potem na tekmovanju taka rešitev vredna 20 točk, a bo popravljavec po lastni presoji lahko dodatno odbil točke zaradi stila pisanja. Vse spremenljivke v kodi so enočrkovne, prav tako pa noben del rešitve ni utemeljen. Če bi imel tekmovalec v kodi napako ali bi kakšen del rešitve napisal z nejasno oznako, bi lahko zaradi pomanjkanja razumevanja popravljavec taki rešitvi dal mnogo manj točk, saj nebi bilo razvidno, da tekmovalec nalogo sploh razume.

```
preberemo n
preberemo x
nastavimo v=1
nastavimo p=x

ponavljamo (n-1)-krat
    preberemo x
    če je x < p
        v=v+1
        p=x

    če ni
        izpišemo v
        v=1
        p=x

izpišemo v
```

2.2 Palindromski oklepaji

Ključna opazka pri reševanju naloge je, da niz ne mora biti hkrati zrcalen in palindrom. Če je niz palindrom, mora namreč prvi znak biti enak zadnjemu, če pa je zrcalen, pa mora biti prvi znak zrcalna slika zadnjega. Navodila popravljanja za to opazko predvidijo 6 točk, tudi če rešitev ne reši naloge. Nadaljnje reševanje lahko razdelimo na dva neodvisna dela — preverjanje, ali je niz palindrom, in preverjanje, ali je niz zrcalen.

Spodnji program preveri le, če je niz palindrom, tekmovalec pa je dodal še opombo, da je niz lihe dolžine vedno bodisi palindrom bodisi navaden. Takšna rešitev bi na tekmovanju dobila 13 točk.

```
#include <stdio.h>

// Predpostavimo, da je niz dolg največ 1000 znakov
char niz[1001];
```

```c
char obrnjen[1001];

int main() {
    scanf("%[^\n]", niz);

    // Opazimo, da niz ne more biti hkrati palindrom in zrcalen,
    // saj bi potem prvi znak moral biti hkrati enak in različen
    // od zadnjega znaka. Zato lahko le posebej preverimo, če je
    // niz palindrom ali če je zrcalen

    // Za preverjanje palindromskosti bomo obrnili niz, potem
    // pa ga primerjali z originalnim
    int dolzina = strlen(niz);
    for (int i = 0; i < dolzina; i++) {
        obrnjen[i] = niz[dolzina-i-1];
    }

    if (strcmp(niz, obrnjen) == 0) {
        printf("palindrom\n");
    }

    // ne znam preveriti, če je niz zrcalen, kratek razmislek pa
    // pove, da niz lihe dolžine ne more biti nikoli zrcalen, saj
    // znak na sredini ne mora biti enak svoji zrcalni sliki.
    if (dolzina % 2 == 1) {
        printf("navaden\n");
    } else {
        printf("zrcalen izraz\n");
    }
    return 0;
}
```

V spodnji rešitvi tekmovalec pravilno določi, če je niz palindrom in če je zrcalen, a manjka razmislek o tem, da niz ne more biti hkrati palindrom in zrcalen, za kar odbijemo nekaj točk. Poleg tega je tekmovalec pozabil prebrati niz in namesto `zrcalen izraz` izpisal le `zrcalen`, za kar dodatno odbijemo dve točki. Tekmovalec prejme 20 točk.

```c
#include <stdio.h>

char niz[300];
char obrnjen[300], zrcalen[300];

```

```
int main() {
    int dolzina = strlen(niz);

    // Prvo izračunamo obrnjen in zrcalen niz
    for (int i = 0; i < dolzina; i++) {
        obrnjen[i] = niz[dolzina-i-1];

        // tudi pri zrcaljenju se niz obrne, zato lahko gledamo
        // obrnjen
        if (obrnjen[i] == '(') zrcalen[i] = ')';
        if (obrnjen[i] == ')') zrcalen[i] = '(';
        if (obrnjen[i] == '[') zrcalen[i] = ']';
        if (obrnjen[i] == ']') zrcalen[i] = '[';
    }

    // preverimo, če se niz ujema z obrnjenim in zrcalnim
    if (strcmp(niz, obrnjen) == 0 && strcmp(niz, zrcalen)) {
        printf("palindrom in zrcalen\n");
    } else if (strcmp(niz, obrnjen) == 0) {
        pritnf("palindrom\n");
    } else if (strcmp(niz, zrcalen) == 0) {
        printf("zrcalen\n");
    } else {
        printf("navaden\n");
    }

    return 0;
}
```