Latvijas informātikas olimpiāde
Svarīga informācija dalībai olimpiādē

Latvijas informātikas olimpiāde

Saturs

1. Svarīgi zināt

1.1. Vispārīgi

1. Datu ievadei/izvadei jālieto standarta ievads/izvads (Skat. Datu ievads un izvads). Programmai sacensību sistēmā nav atļauts veidot datnes un strādāt ar tām.
2. Programmēšanas valodu dokumentācija ir pieejama sadaļā «Dokumentācija».
3. Sadaļā «Testēšana» var pārbaudīt risinājuma darbību sacensību sistēmā ar savu ievadu. Izvada izmēra ierobežojums ir 100 KiB.
4. Java, Python u.c. valodās nenulles atgriešanās kodu var izraisīt arī neapstrādāta izņēmumsituācija (piemēram, piekļūšana ārpus masīva robežām).
5. Sacensību sistēmā izmantotās programmēšanas vides un parametrus skatīt: https://contest.lio.lv/
6. Punktu skaitu par uzdevumu nosaka hronoloģiski pēdējais iesūtītais risinājums, kas atzīts par derīgu testēšanai (kompilējas). Precīzāku informācijas skatīt sacensību norises kārtības dokumentā.
7. Dalībnieka pienākums ir katra uzdevuma hronoloģiski pēdējo pamattestēšanai derīgo risinājumu saglabāt darba datorā.

1.2. C++

1. C/C++ long long skaitļu lasīšanai un rakstīšanai izmantojot scanf, printf funkcijas jālieto %lli vai %lld, nevis %I64i vai %I64d.

1.3. Java

1. Java programmās nav jālieto «package» deklarācija.
2. Valodā Java rakstītam risinājumam ir jālieto klases vārds, kas ir definēts uzdevuma formulējuma sadaļā «Ierobežojumi un prasības».

1.4. Go

1. Go dokumentācija:
   - Uz sava datora palaiž `godoc -http=localhost:6060` komandu
   - Pārlūkā atver http://localhost:6060/
   - Ja godoc komanda nav pieejama, tad vērsieties pie datorklases administratora, lai uzstāda godoc dokumentāciju izmantojot komandu `go install golang.org/x/tools/cmd/godoc`.

1.5. Python

1. Sacensību sistēmā ir pieejamas divas Python 3 implementācijas:
   - «CPython» ir visplašāk izmantotā python implementācija.
   - «PyPy» ir populāra alternatīva Python implementācija, kura var strādāt ātrāk nekā CPython implementācija.
2. Lai noskaidrotu Python programmas nenulles atgriešanās koda iemeslu versiju nesaderības dēļ:
   1. Iekļaujiet programmu try, except blokā (Skat. piemēru)
   2. Iesūtiet programmu sacensību sistēmas „Testēšana“ sadaļā norādot savu programmu un ievaddatus.
   3. Lejupielādējiet izpildes izvaddatus. Izlasiet izvadīto kļūdas iemeslu.

   Piemēra programma:

```
try:
    # Jūsu python programma
    d = {"abols": 5} | {"kaposts": 4}
    print(d)
except Exception as e:
    print(e)
    # Izvaddatos tiks izdrukāts:
    # unsupported operand type(s) for |: 'dict' and 'dict'
```

2. Datu ievads un izvads

2.1. Standarta ievads un izvads

2.1.1. Ievads

Par standarta ievadu/izvadu sauc datu plūsmas, kas, programmai sākot darbu, jau ir atvērtas un gatavas darbam. Tas atšķiras no datņu plūsmām, kas programmētājam ir jāatver pašam, norādot datnes nosaukumu un ceļu. To, ar ko programmas standarta ievads/izvads ir saistīti (datnes, starpprocesu saziņa utt.), nosaka vide, kurā programma tiek palaista, un veids, kā tas tiek darīts. Pašai programmai par to nav jāuztraucas — pietiek zināt, ka ar standarta ievadu/izvadu var strādāt kā ar parastām datu plūsmām.

Pastāv trīs standarta datu plūsmas (iekavās doti plūsmu nosaukumi C, C++, Pascal, Java):

- Standarta ievads (stdin, std::cin, input, System.in) — datu plūsma, no kuras programma nolasa ievaddatus.
- Standarta izvads (stdout, std::cout, output, System.out) — datu plūsma, kurā programma ieraksta izvaddatus.
- Standarta diagnostiskais izvads (stderr, std::cerr/std::clog, errout, System.err) — datu plūsma, kurā programma izvada kļūdas un diagnostiskos paziņojumus. To var lietot programmu atkļūdošanai.

Vairāk par standarta ievadu/izvadu var izlasīt Wikipedia.

2.1.2. Programmu piemēri

Tālāk ir doti piemēri darbam ar standarta plūsmām dažādās programmēšanas valodās (bez kļūdu apstrādes). Uzdevums ir nolasīt divus skaitļus un izvadīt to summu.

2.1.2.1. C

```c
#include <stdio.h>

int main()
{
    int a, b;

    scanf("%d %d", &a, &b);
    printf("%d\n", a + b);
}
```

2.1.2.2. C++

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a, b;

    cin >> a >> b;
    cout << a + b << "\n";
}
```

2.1.2.3. Pascal

```
program Sum;

var
    a, b: integer;

begin
    readln(a, b);
    writeln(a + b);
end.
```

2.1.2.4. Python 3

```
a, b = map(int, input().split())
print(a + b)
```

2.1.2.5. Java

```java
import java.io.*;
import java.util.*;

public class Sum {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        PrintWriter out = new PrintWriter(System.out);

        int a = in.nextInt(), b = in.nextInt();
        out.println(a + b);

        out.flush();
    }
}
```

2.1.2.6. Go

```go
package main

import (
    "bufio"
    "fmt"
    "os"
)

func main() {
    var a, b int
    in := bufio.NewReader(os.Stdin)
    out := bufio.NewWriter(os.Stdout)

    fmt.Fscanf(in, "%d %d\n", &a, &b)
    fmt.Fprintf(out, "%d\n", a+b)
    out.Flush()
}
```

Piemērs izmanto bufio pakotni ievada, izvada buferizācijai.

2.1.3. Veiktspējas uzlabojumi

Piezīme: šajā nodaļā dotie padomi ir paredzēti pieredzējušiem lietotājiem.

Ja ievaddatu vai izvaddatu apjoms ir ļoti liels, ievads un izvads var aizņemt ievērojamu laiku. Ir daži veidi, kā to samazināt:

- C++ ieteicams izsaukt `std::ios::sync_with_stdio(false);` programmas sākumā, ja nav paredzēts lietot C (`printf()`/`scanf()`) funkcijas.
- C++ ieteicams izsaukt `std::cin.tie(NULL);` programmas sākumā, ja uzdevums nav interaktīvs.
- C++ ieteicams lietot `'\n'`, nevis `std::endl`, ja uzdevums nav interaktīvs.
- Java `Scanner` vietā ieteicams lietot `StringTokenizer` vai `StreamTokenizer`, kopā ar `BufferedReader`.
- Java, lietojot `PrintWriter`, ieteicams izvadīt vērtības pa vienai un nelietot `PrintWriter.printf()` funkciju.
- Programmēšanas valodā Python datu ielasīšanai iesakām izmantot standarta ievada `input` funkciju:

  ```python
  import sys
  input = sys.stdin.readline
  ```

Lielu datu ievadu ārpus sacensībām var izmēģināt CodeChef INTEST uzdevumā.

2.2. Darbs komandrindā

Piezīme: Windows operētājsistēmā komandrindas logu var atvērt, aizejot uz risinājuma mapi, spiežot *Shift+labā peles poga* uz mapes fona un uzvēloties „Open command window here“.

Piezīme: Unix-veidīgu operētājsistēmu lietotājiem, kā arī PowerShell lietotājiem piemēros ir jāaizvieto `prog` ar `./prog`.

Palaižot programmu komandrindā, programmu standarta plūsmas pēc noklusējuma tiek piesaistītas attiecīgajam terminālim. Lietotājs var ievadīt datus programmā un iegūt rezultātus. Piemēram:

```
$ prog                  ⇦ Lietotājs ievada programmas nosaukumu "prog" un palaiž programmu ar Enter
1 2                     ⇦ Lietotājs ievada datus (kopā ar Enter)
3                       ⇦ Programma izvada atbildi
$ ▉                     ⇦ Komandrinda gaida nākamo lietotāja ievadīto komandu
```

Standarta ievadu/izvadu var novirzīt datnēs. Piemēram, var piesaistīt programmas standarta ievadu datnei:

```
$ prog < in.txt         ⇦ Lietotājs palaiž programmu un standarta ievadā iedod in.txt (kas satur "1 2")
3                       ⇦ Programma nolasa ievadu un izvada rezultātu
$ ▉                     ⇦ Komandrinda gaida nākamo lietotāja ievadīto komandu
```

Kā arī var piesaistīt datnes vienlaikus gan standarta ievadam, gan izvadam:

```
$ prog < in.txt > out.txt   ⇦ Lietotājs palaiž programmu, ievadā iedod in.txt, bet izvadu novirza out.txt
$ ▉                         ⇦ Terminālī nekas netiek izvadīts, bet out.txt tagad satur "3"
```

Vairāk par pāradresāciju var izlasīt Wikipedia.

2.3. Datu ievads, izvads interaktīvos (komunikācijas) uzdevumos

2.3.1. Ievads

Parasti sacensību programmēšanas uzdevumos tiek padoti nemainīgi ievaddati, un pēc visu ievaddatu ielasīšanas programma izvada atbildi. Savukārt interaktīvajos uzdevumos programmai ir jāveido divvirzienu komunikācija ar vērtēšanas sistēmu (interaktoru). Komunikācijas uzdevumos programma var izvadīt datus uz standarta izvadu, ko var nolasīt interaktors, programma var nolasīt datus no standarta ievada, ko interaktors padod programmai, piemēram, atbildot uz programmas izvadītajiem datiem standarta izvadā.

Interaktīvajos uzdevumos īpaša uzmanība ir jāpievērš tam, lai dati, ko programma padod izvadīšanai uz standarta izvadu, tiešām sasniedz standarta izvadu un tālāk jau interaktoru. Lai ar lielu veiktspēju varētu ielasīt, izvadīt datus, mūsdienās programmas izmanto papildus datu apgabalu (*buferi/masīvu*), kurā uzkrāt datus, lai varētu ielasīt, izvadīt lielāku apjomu ar datiem vienā piegājienā, jo datu ielasīšana, izvadīšana no programmas ir relatīvi resursietilpīga operācija.

Tādēļ, lai pārliecinātos, ka visi dati no *bufera* tiek izvadīti un ir nodoti interaktoram pirms, piemēram, tiek gaidīta atbilde no tā, ir jāveic manuāla izvada sinhronizācija (*flush*).

Izvada sinhronizācija (*flush*) tiek izpildīta, piemēram, pēc šādu programmas izteikumu izpildes:

Programmēšanas valoda	**Izteikums**	**Komentārs**
C	`fflush(stdout);`	
C++	`std::cout << std::endl;`	Iesāk jaunu rindu (izvada jaunas rindas simbolus) un veic izvada sinhronizāciju.
	`std::cout << std::flush;`	
	`std::cout.flush();`	
Python3	`print("Hello world!", flush=True)`	Izvada rindu ar *Hello world!* un veic izvada sinhronizāciju.
	`sys.stdout.flush()`	
Pascal	`flush(output);`	
Go	`fmt.Println(K)`	Komanda izvada rindu ar vērtību K un sinhronizē izvadu.
	`writer.Flush()`	Veic *bufera* sinhronizāciju, ja tiek izmantota `bufio` bibliotēka. `writer := bufio.NewWriter(os.Stdout)` Izvadot ar `fmt.Fprintln(writer, k)` netiek veikta izvada sinhronizācija.
Java	`System.out.flush();`	
	`out.flush();`	Sinhronizācija, ja izmanto `PrintWriter`. `out = new PrintWriter(System.out);`

2.3.2. Programmu piemēri

Uzdevuma piemērs

Apraksts

Dators „ir iedomājies“ naturālu skaitli X robežās no 1 līdz N.

Skaitļa X atminēšanai Jūsu programma var veikt *vaicājumus*. Katrs vaicājums ir formā „Vai iedomātais skaitlis ir K?“, kur $1 \leq K \leq N$, un uz katru šādu vaicājumu dators dod vienu no trim atbildēm:

- 1, ja $K < X$,
- 0, ja $K = X$,
- -1, ja $K > X$.

Skaitlis X ir uzminēts **tikai tad**, ja ir izdarīts vaicājums uz kuru saņemta atbilde 0.

Uzrakstiet datorprogrammu, kas atrod skaitli X.

Komunikācija

Šis ir interaktīvs uzdevums. Jūsu programmai, sākot darbu, pirmā ievada rinda satur veselu skaitli N ($1 \leq N \leq 500$). Iedomātā skaitļa vērtību X vērtēšanas sistēma tur slepenībā. Tad jūsu programma var veikt vaicājumus, izvadā rakstot vērtību K ($1 \leq K \leq N$). Vērtēšanas sistēma izdod atbildi nākamajā ievada rindā. Atbilde ir vesels skaitlis – -1, 0 vai 1, kā aprakstīts iepriekš. Kad uz vaicājumu tiek izdota atbilde 0, jūsu programmai darbs jābeidz.

Tālāk ir doti piemēri darbam ar standarta plūsmām dažādās programmēšanas valodās, lai veiktu komunikāciju atbilstoši dotā uzdevuma aprakstam.

2.3.2.1. C

```c
#include <stdio.h>

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

    int k = n / 2 + 1;
    while (true) {
        printf("%d\n", k);
        fflush(stdout);

        int reply;
        scanf("%d", &reply);
        if (reply == -1) {
          k = k - 1;
        } else if (reply == 1) {
          k = k + 1;
        } else {
          break;
        }
    }
}
```

2.3.2.2. C++

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;

    int k = n / 2 + 1;
    while (true) {
        cout << k << "\n";
        cout.flush();

        int reply;
        cin >> reply;
        if (reply == -1) {
          k = k - 1;
        } else if (reply == 1) {
          k = k + 1;
        } else {
          break;
        }
    }
}
```

2.3.2.3. Pascal

```pascal
program NumberGuessingGame;

var
  n, k, reply: Integer;

begin
  readln(n);
  k := n div 2 + 1;

  while True do
  begin
    writeln(k);
    flush(output);
    readln(reply);

    if reply = -1 then
      k := k - 1
    else if reply = 1 then
      k := k + 1
    else
      break;
  end;
end.
```

2.3.2.4. Python 3

```python
import sys

n = int(input())
k = n // 2 + 1

while True:
    print(k)
    sys.stdout.flush()

    reply = int(input())

    if reply == -1:
        k -= 1
    elif reply == 1:
        k += 1
    else:
        break
```

2.3.2.5. Java

```java
import java.util.Scanner;
import java.io.PrintWriter;

public class NumberGuessingGame {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        PrintWriter out = new PrintWriter(System.out);

        int n = in.nextInt();
        int k = n / 2 + 1;

        while (true) {
            out.println(k);
            out.flush();

            int reply = in.nextInt();

            if (reply == -1) {
                k = k - 1;
            } else if (reply == 1) {
                k = k + 1;
            } else {
                break;
            }
        }
    }
}
```

2.3.2.6. Go

```go
package main

import (
  "bufio"
  "fmt"
  "os"
)

func main() {
  reader := bufio.NewReader(os.Stdin)
  writer := bufio.NewWriter(os.Stdout)

  var n, k, reply int
  fmt.Fscan(reader, &n)
  k = n/2 + 1

  for {
    fmt.Fprintln(writer, k)
    writer.Flush()

    fmt.Fscan(reader, &reply)

    if reply == -1 {
      k = k - 1
    } else if reply == 1 {
      k = k + 1
    } else {
      break
    }
  }
}
```

Piemērs izmanto bufio pakotni ievada, izvada buferizācijai.

2.3.3. Darbs komandrindā interaktīvos uzdevumos

Kā vispārīgi strādāt komandrindā lasiet sadaļā „Darbs komandrindā“.

Palaižot interaktīvu programmu komandrindā interaktora lomu varat uzņemties jūs ievadot ievaddatus komandrindas skatā, apstrādājot programmas izvadu.

Piemēram, iepriekš dotajā piemēra uzdevumā interaktoru var simulēt sekojoši:

1. Jūs iedomājieties, ka $N = 10$ un iedomātais skaitlis ir 7.
2. Jūs izsaucat programmu komandrindā.

```
$ prog                  ⇦ Lietotājs ievada programmas nosaukumu "prog" un palaiž programmu ar Enter
                        ⇦ Programma gaida ievaddatus
```

3. Jūs padodat programmai skaitli N un programma jums izdod vaicājumu.

```
$ prog                  ⇦ Lietotājs ievada programmas nosaukumu "prog" un palaiž programmu ar Enter
10                      ⇦ Jūs ievadījāt vērtību N
6                       ⇦ Programma izvadīja vaicājumu vērtībai 6
                        ⇦ Programma gaida ievaddatus
```

4. Jūs padodat programmai atbildi uz programmas izdarīto vaicājumu.

```
$ prog                  ⇦ Lietotājs ievada programmas nosaukumu "prog" un palaiž programmu ar Enter
10                      ⇦ Jūs ievadījāt vērtību N
6                       ⇦ Programma izvadīja vaicājumu vērtībai 6
1                       ⇦ Jūs ievadījāt vērtību 1, jo jūsu iedomātā vērtība bija lielāka nekā programmas vaicātā
7                       ⇦ Programma izvadīja vaicājumu vērtībai 7
                        ⇦ Programma gaida ievaddatus
```

5. Jūs padodat programmai atbildi uz programmas izdarīto vaicājumu.

```
$ prog                  ⇦ Lietotājs ievada programmas nosaukumu "prog" un palaiž programmu ar Enter
10                      ⇦ Jūs ievadījāt vērtību N
6                       ⇦ Programma izvadīja vaicājumu vērtībai 6
1                       ⇦ Jūs ievadījāt vērtību 1, jo jūsu iedomātā vērtība bija lielāka nekā programmas vaicātā
7                       ⇦ Programma izvadīja vaicājumu vērtībai 7
0                       ⇦ Jūs ievadījāt vērtību 0, jo programma uzminēja jūsu iedomāto vērtību
$                       ⇦ Programma pabeidza darbību un komandrinda gaida nākamo izpildāmo komandu
```

6. Programma pabeidza darbību.

Šādi izmantojot komandrindu jūs varat pārbaudīt jūsu programmas darbību.

2.4. Datu ievads valodā Java

2.4.1. Vienkāršs ievads un izvads

Visvienkāršākā klase datu ievadam ir `Scanner`. Ar tās palīdzību var uzreiz lasīt dažādu tipu datus. Izvadam var izmantot `System.out`. Diemžēl, abas šīs klases nenodrošina pietiekami ātru datu ievadu un izvadu un sacensību apstākļos lielāka apjoma datiem var izrādīties par lēnu.

```
import java.util.Scanner;

public class IOTest {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        System.out.println(a + b);
        System.out.flush();
    }
}
```

2.4.2. Ātrs ievads un izvads

Ātram datu ievadam ieteicams izmantot klasi `StreamTokenizer`. Šī klase automātiski sadala ievadu skaitļu un teksta daliņās. Lai nolasītu kārtējo daļiņu, jāizsauc komanda `nextToken()`. Skaitļu un teksta vērtības tiek saglabātas attiecīgi laukos `nval` un `sval`.

Ātram datu izvadam izmantojiet klasi `PrintWriter`. Nav ieteicams lietot tās metodi `printf()`, kas var strādāt lēni (izmantojiet `print()` vai `println()`). Tāpat nav ieteicams izvadīt vairākos mainīgos reizē, kas arī var palēlināt izvadu (piemēram, `out.println(a + " " + b)` vietā labāk lietot `out.print(a); out.print(" "); out.println(b);`).

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

public class IOTest {
    static StreamTokenizer in;
    static PrintWriter out;

    static int nextInt() throws IOException {
        in.nextToken();
        return (int) in.nval;
    }

    static String nextString() throws IOException {
        in.nextToken();
        return in.sval;
    }

    public static void main(String[] args) throws IOException {
        in = new StreamTokenizer(new BufferedReader(new java.io.InputStreamReader(System.in)));
        out = new PrintWriter(System.out);
        int a = nextInt();
        int b = nextInt();
        out.println(a + b);
        String s = nextString();
        out.println(s);
        out.flush();
    }
}
```

2.4.3. Long tipa skaitļu lasīšana

Skaitliskās vērtības `StreamTokenizer` glabā `double` tipa laukā `nval`, kura precizitātes nepietiek, lai saglabātu `long` tipa skaitļus. Lai nolasītu šādus skaitļus, var nodefinēt zīmes un ciparus kā tekstuālas vērtības priekš `StreamTokenizer`. Tad skaitli var nolasīt kā tekstu un nopārsēt iegūto virkni par skaitli. Jāņem vērā, ka šajā gadījumā visi skaitļi tiks lasīti kā teksts (arī `int` lieluma skaitļi).

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StreamTokenizer;
public class IOTest {
    static StreamTokenizer in;
    static PrintWriter out;

    static Long nextLong() throws IOException {
        in.nextToken();
        return Long.parseLong(in.sval);
    }

    public static void main(String[] args) throws IOException {
        in = new StreamTokenizer(new BufferedReader(new java.io.InputStreamReader(System.in)));
        in.resetSyntax();
        in.whitespaceChars(0, 32);
        in.wordChars('0', '9');
        in.wordChars('-', '-');
        out = new PrintWriter(System.out);
        long a = nextLong();
        long b = nextLong();
        out.println(a + b);
        out.flush();
    }
}
```

2.4.4. Citas metodes

Dažreiz vēl ātrāk par `StreamTokenizer` strādā pati `BufferedReader` klase, un citos gadījumos var būt efektīvāk pielietot `StringTokenizer` klasi. Par šiem rīkiem sīkāk var izlasīt Java dokumentācijā. Tomēr `StreamTokenizer` ātrdarbība parasti ir pietiekama visos gadījumos.