PJISP zbirka zadataka

Petar Marić
Srđan Popov

nov 07, 2020

CONTENTS

CHAPTER

1

UVOD

Ova zbirka rešenih zadataka namenjena je studentima Fakulteta tehničkih nauka Univerziteta u Novom Sadu, kao pomoćni udžbenik za predmet „Programski jezici i strukture podataka". Svi primeri rešenja napisani su u programskom jeziku C, čije poznavanje je potrebno za upotrebu zbirke.

Zbirka se može kupiti u papirnom obliku ili besplatno čitati na web sajtu http://pjisp.petarmaric.com/zbirka-zadataka.

Ovo delo je u celosti bazirano na tehnologijama otvorenog koda, kao što su Python, Sphinx, Docker i Fabric. Izvorni kod ovog dela dostupan je na https://github.com/petarmaric/pjisp-zbirka-zadataka pod licencom otvorenog koda, pogledati glavu *Licenca* za detaljne informacije o uslovima licence.

CHAPTER

2

PRIMERI ZADATAKA ZA T12

2.1 Zadatak „Transformisati niz (bez pomoćnog)”

Autor zadatka: Srđan Popov <srdjanpopov@uns.ac.rs>

Unosom sa tastature je zadat niz `A`, neoznačenih celih brojeva od maksimalno 30 elemenata. Transformisati niz (bez pomoćnog) tako da se svaki element niza pojavi samo jednom.

Na izlazu štampati broj elemenata zadatog niza, elemente zadatog niza, broj elemenata transformisanog niza i elemente niza.

2.1.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 30

int main()
{
    int A[MAX_SIZE];
    int n;
    int i,j,k;
    do
    {
        printf("Unesite broj elemenata niza:");
        scanf("%d", &n);
    } while(n<=1 || n > MAX_SIZE);
```

```
    for(i=0; i<n; i++)
    {
        printf("A[%d]=", i);
        scanf("%d", &A[i]);
    }
    printf("\n");
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
            if(A[i]==A[j])
            {
                n--;//smanjuje se broj elemenata niza
                for(k=j;k<n;k++)
                    A[k]=A[k+1];
                j--;//novi element na j-tom indeksu se mora ponovo testirati
            }
    for(i=0; i<n; i++)
    {
        printf("\nA[%d]=%d", i,A[i]);
    }
    printf("\n");
    return 0;
}
```

2.2 Zadatak „Par/nepar”

Autor zadatka: Srđan Popov <srdjanpopov@uns.ac.rs>

Unosom sa tastature je zadat niz `X` od maksimalno 50 celobrojnih elemenata. Učitati `n` elemenata u niz `X` i formirati nizove `A` i `B`, pri čemu su elementi niza `A` parni elementi niza `X`, a elementi niza `B` su elementi niza `X` sa neparnim indeksom (1,3,5,. . .).

Izračunati `SRVA` srednju vrednost niza `A`. Na izlazu štampati elemente nizova `A`, `B` i `SRVA` srednju vrednost niza `A`.

2.2.1 Primer rešenja

```
#include <stdio.h>

#define MAX_SIZE 30

int main()
{
    int X[MAX_SIZE];
    int A[MAX_SIZE];
    int B[MAX_SIZE/2];
    int n;
    int i;
    int j=0,k=0,sum=0;
    double SRVA;
    do
    {
```

```
        printf("Unesite broj elemenata niza:");
        scanf("%d", &n);
    } while(n<=1 || n > MAX_SIZE);

    for(i=0; i<n; i++)
    {
        printf("X[%d]=", i);
        scanf("%d", &X[i]);
    }
    for(i=0;i<n;i++)
    {
        if(X[i]%2==0) A[j]=X[i], j++;//parni elementi
        if(i%2!=0) B[k]=X[i], k++;//neparni indeks
    }

    for(i=0; i<j; i++)
    {
        sum+=A[i];
    }
    SRVA=(double)sum/j;

    printf("\n\n");
    puts("Elementi niza A:");
    for(i=0; i<j; i++)
    {
        printf("\nA[%d]=%d", i, A[i]);
    }
    printf("\n\n");

    puts("Elementi niza B:");
    for(i=0; i<k; i++)
    {
        printf("\nB[%d]=%d", i, B[i]);
    }
    printf("\n\n");

    printf("\nSRVA=%.2lf\n\n",SRVA);

    return 0;
}
```

2.3 Zadatak „Obrnuti redosled elemenata”

Autor zadatka: Srđan Popov <srdjanpopov@uns.ac.rs>

Dat je celobrojni niz od maksimalno 30 elemenata. Obrnuti redosled elemenata u nizu (bez upotrebe pomoćnog niza) i zatim odštampati rezultat.

Za sledeće ulazne podatke `[3, 7, 4, 1, 5]` rešenje je `[5, 1, 4, 7, 3]`.

2.3.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 30

int main()
{
    int A[MAX_SIZE];
    int pom;
    int n;
    int i;
    do {
        printf("Unesite broj elemenata niza:");
        scanf("%d", &n);
    } while(n<=1 || n > MAX_SIZE);

    for(i=0; i<n; i++)
    {
        printf("A[%d]=", i);
        scanf("%d", &A[i]);
    }
    for(i=0;i<=n/2;i++)
    {
        pom=A[i];
        A[i]=A[n-1-i];
        A[n-1-i]=pom;
    }
    for(i=0; i<n; i++)
    {
        printf("\nA[%d]=%d", i,A[i]);
    }
    printf("\n");
    return 0;
}
```

2.4 Zadatak „Vrednosti veće od srednje”

Autor zadatka: Srđan Popov <srdjanpopov@uns.ac.rs>

Unosom sa tastature je zadat niz X od maksimalno 30 celobrojnih elemenata. Naci broj elemenata niza cija je vrednost veca od srednje vrednosti niza. Na izlazu štampati niz X, srednju vrednost niza (SRVX) i broj elemenata niza cija je vrednost veca od srednje vrednosti niza.

2.4.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 30

int main()
{
```

```
    int X[MAX_SIZE];
    int n,i;
    int sum=0;
    int br=0;
    double SRVX;
    do {
        printf("Unesite broj elemenata niza:");
        scanf("%d", &n);
    } while(n<=1 || n > MAX_SIZE);

    for(i=0; i<n; i++) {
        printf("X[%d]=", i);
        scanf("%d", &X[i]);
    }

    for(i=0; i<n; i++) {
        sum+=X[i];
    }
    SRVX=(double)sum/n;

    for(i=0;i<n;i++)
        if(X[i]>SRVX) br++;

    printf("\n\n");
    puts("Elementi niza X:");
    for(i=0; i<n; i++) {
        printf("\nX[%d]=%d", i, X[i]);
    }
    printf("\n\n");

    printf("\nSRVX=%.2lf\n\n",SRVX);

    printf("\nBroj elemenata niza X vecih od srednje vrednosti je:%d\n\n",br);

    return 0;
}
```

2.5 Zadatak „Brojevne sredine niza”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz `A` od maksimalno 10 celobrojnih elemenata. Učitati `n` elemenata a zatim:

- ispisati sadržaj celokupnog niza, od poslednjeg elementa ka početnom
- odrediti i ispisati kvadratnu sredinu celokupnog niza (`KS`)
- odrediti i ispisati aritmetičku sredinu elemenata niza koji su na neparnim indeksima (`AS`)

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`
- sve realne promenljive trebaju biti tipa `double`

- sve realne brojeve ispisivati zaokružene na 2 decimale

Za sledeće ulazne podatke:

```
n = 5
A = {4, 2, 6, -7, 1}
```

očekivani izlaz je u sledećem formatu:

```
A[4] = 1
A[3] = -7
A[2] = 6
A[1] = 2
A[0] = 4

KS = 4.60
AS = -2.50
```

Korisne matematičke formule:

- kvadratna sredina niza $KS = \sqrt{\frac{1}{n} \sum {x_i}^2}$
- aritmetička sredina niza $AS = \frac{1}{n} \sum x_i$

2.5.1 Primer rešenja

```c
#include <stdio.h>
#include <math.h>

#define MAX_SIZE 10

int main() {
    int A[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite %d. clan niza: ", i+1);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=n-1; i>=0; i--) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    double KS_sum = 0;
    for(i=0; i<n; i++) {
        KS_sum += pow(A[i], 2);
    }
```

```
    double KS = sqrt(KS_sum / n);
    printf("\nKS = %.2lf\n", KS);

    double AS_sum = 0;
    int AS_count = 0;
    for(i=1; i<n; i+=2) {
        AS_sum += A[i];
        AS_count++;
    }
    double AS = AS_sum / AS_count;
    printf("AS = %.2lf\n", AS);

    return 0;
}
```

2.6 Zadatak „Max razlika”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz A od maksimalno 15 celobrojnih elemenata. Učitati n elemenata a zatim naći 2 elementa čija je razlika najveća. Sprovesti zaštitu unosa nad podatkom n. Ako se vrši sortiranje, svejedno je koji će se algoritam koristiti.

Za sledeće ulazne podatke:

```
n = 5
A = {4, 2, 6, -7, 1}
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 4
A[1] = 2
A[2] = 6
A[3] = -7
A[4] = 1

max = 6 - -7 = 13
```

2.6.1 Primer rešenja

```
#include <stdio.h>

#define MAX_SIZE 15

int main() {
    int A[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);
```

```
    int i;
    for(i=0; i<n; i++) {
        printf("Unesite %d. clan niza: ", i+1);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    int min = A[0];
    int max = A[0];
    for(i=1; i<n; i++) {
        if(A[i] < min) {
            min = A[i];
        } else if (A[i] > max) {
            max = A[i];
        }
    }

    printf("\nmax = %d - %d = %d", max, min, max-min);

    return 0;
}
```

2.7 Zadatak „Max zbir”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz A od maksimalno 15 celobrojnih elemenata. Učitati n elemenata a zatim naći 2 elementa čiji je zbir najveći. Sprovesti zaštitu unosa nad podatkom n. Ako se vrši sortiranje, svejedno je koji će se algoritam koristiti.

Za sledeće ulazne podatke:

```
n = 5
A = {4, 2, 6, -7, 1}
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 4
A[1] = 2
A[2] = 6
A[3] = -7
A[4] = 1

max = 6 + 4 = 10
```

2.7.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 15

int main() {
    int A[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite %d. clan niza: ", i+1);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    int j, tmp;
    for(i=0; i<n-1; i++) {
        for(j=i+1; j<n; j++) {
            if (A[i] < A[j]) {
                tmp = A[i];
                A[i] = A[j];
                A[j] = tmp;
            }
        }
    }

    printf("\nmax = %d + %d = %d", A[0], A[1], A[0] + A[1]);

    return 0;
}
```

2.8 Zadatak „Zbir kvadrata dvocifrenih brojeva”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz A od maksimalno 15 dvocifrenih prirodnih brojeva. Učitati n elemenata a zatim proveriti da li postoje parovi elemenata za koje važi da je zbir kvadrata njihovih cifara jednak. Sprovesti zaštitu unosa nad podatkom n.

Za sledeće podatke:

```
n = 7
A = {42, 67, 12, 36, 76, 29, 21}
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 42
A[1] = 67
A[2] = 12
A[3] = 36
A[4] = 76
A[5] = 29
A[6] = 21

zbir(A[0]) = 20
zbir(A[1]) = 85
zbir(A[2]) = 5
zbir(A[3]) = 45
zbir(A[4]) = 85
zbir(A[5]) = 85
zbir(A[6]) = 5

zbir(A[1]) == zbir(A[4]) == 85
zbir(A[1]) == zbir(A[5]) == 85
zbir(A[2]) == zbir(A[6]) == 5
zbir(A[4]) == zbir(A[5]) == 85

count = 4
```

2.8.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 15

int main() {
    int A[MAX_SIZE];
    int kvadrati[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite %d. clan niza: ", i+1);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        int cif_des = A[i] / 10;
        int cif_jed  = A[i] % 10;
```

```
        kvadrati[i] = cif_des * cif_des + cif_jed * cif_jed;

        printf("zbir(A[%d]) = %d\n", i, kvadrati[i]);
    }

    int j;
    int count = 0;
    printf("\n");
    for(i=0; i<n-1; i++) {
        for(j=i+1; j<n; j++) {
            if (kvadrati[i] == kvadrati[j]) {
                printf("zbir(A[%d]) == zbir(A[%d]) == %d\n", i, j, kvadrati[i]);
                count++;
            }
        }
    }

    printf("\ncount = %d\n", count);

    return 0;
}
```

2.9 Zadatak „Neparni susedi”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz `A` od maksimalno 10 pozitivnih celobrojnih elemenata. Učitati `n` elemenata a zatim:

- ispisati sadržaj celokupnog niza
- proveriti da li postoje parovi *susednih* elemenata za koje važi da su neparni

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`

Za sledeće ulazne podatke:

```
Unesite broj elemenata niza (max 10): 8

Unesite A[0]: 5
Unesite A[1]: 3
Unesite A[2]: 0
Unesite A[3]: 4
Unesite A[4]: 3
Unesite A[5]: 7
Unesite A[6]: 1
Unesite A[7]: 15
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 5
A[1] = 3
```

```
A[2] = 0
A[3] = 4
A[4] = 3
A[5] = 7
A[6] = 1
A[7] = 15

odd(A[0]) == odd(A[1]) == 1
odd(A[4]) == odd(A[5]) == 1
odd(A[5]) == odd(A[6]) == 1
odd(A[6]) == odd(A[7]) == 1

count = 4
```

2.9.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 10

int main() {
    int A[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite A[%d]: ", i);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    int count = 0;
    printf("\n");
    for(i=0; i<n-1; i++) {
        if (A[i] % 2 && A[i+1] % 2) {
            printf("odd(A[%d]) == odd(A[%d]) == %d\n", i, i+1, A[i] % 2);
            count++;
        }
    }

    printf("\ncount = %d\n", count);

    return 0;
}
```

2.10 Zadatak „Sumiranje odnazad”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz `A` od maksimalno 10 celobrojnih elemenata. Učitati `n` elemenata a zatim:

- učitati pozitivnu celobrojnu vrednost `x`
- ispisati sadržaj celokupnog niza
- odrediti koliko elemenata (počevši od kraja niza ka početku) je potrebno obuhvatiti da bi njihova suma bila veća od `x`

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`
- smatrati da će suma celokupnog niza `A` uvek biti veća od `x`

Za sledeće ulazne podatke:

```
Unesite broj elemenata niza (max 10): 6
Unesite trazenu vrednost: 4

Unesite A[0]: 6
Unesite A[1]: 5
Unesite A[2]: 4
Unesite A[3]: -3
Unesite A[4]: 2
Unesite A[5]: 1
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 6
A[1] = 5
A[2] = 4
A[3] = -3
A[4] = 2
A[5] = 1

sum(A[5]) == 1
sum(A[4]) == 3
sum(A[3]) == 0
sum(A[2]) == 4
sum(A[1]) == 9

count = 5
```

2.10.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 10

int main() {
```

```
    int A[MAX_SIZE];
    int n;
    int x;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);

    printf("Unesite trazenu vrednost: ");
    scanf("%d", &x);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite A[%d]: ", i);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    int count = 0;
    int sum = 0;
    printf("\n");
    for(i=n-1; i>=0; i--) {
        if (sum > x) {
            break;
        }

        count++;
        sum+=A[i];

        printf("sum(A[%d]) == %d\n", i, sum);
    }

    printf("\ncount = %d\n", count);

    return 0;
}
```

2.11 Zadatak „Slučajni odabir”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz `A` od maksimalno 10 celobrojnih elemenata. Učitati `n` elemenata a zatim:

- učitati celobrojnu vrednost `x`
- ispisati sadržaj celokupnog niza
- odrediti i ispisati verovatnoću da vrednost `x` bude odabrana slučajnim uzorkovanjem iz niza `A` (chance)

Ograničenja:

- sprovesti zaštitu unosa nad podatkom n
- vrednost promenljive chance ispisati na 3 decimale

Za sledeće ulazne podatke:

```
Unesite broj elemenata niza (max 10): 8
Unesite trazenu vrednost: 42

Unesite A[0]: 42
Unesite A[1]: 5
Unesite A[2]: 42
Unesite A[3]: 1
Unesite A[4]: -21
Unesite A[5]: -42
Unesite A[6]: 3
Unesite A[7]: 42
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 42
A[1] = 5
A[2] = 42
A[3] = 1
A[4] = -21
A[5] = -42
A[6] = 3
A[7] = 42

found(A[0]) == 42
found(A[2]) == 42
found(A[7]) == 42

chance = 0.375
```

2.11.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 10

int found(int n) {
    return n % 10;
}

int main() {
    int A[MAX_SIZE];
    int n;
    int x;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);
```

```
    printf("Unesite trazenu vrednost: ");
    scanf("%d", &x);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite A[%d]: ", i);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    int count = 0;
    printf("\n");
    for(i=0; i<n; i++) {
        if (A[i] == x) {
            printf("found(A[%d]) == %d\n", i, A[i]);
            count++;
        }
    }
    double chance = (double)count / n;

    printf("\nchance = %.3lf\n", chance);

    return 0;
}
```

2.12 Zadatak „Apsolutno jednaki”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz `A` od maksimalno 10 celobrojnih elemenata. Učitati `n` elemenata a zatim:

- ispisati sadržaj celokupnog niza
- proveriti da li postoje parovi elemenata za koje važi da su po apsolutnoj vrednosti jednaki

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`

Za sledeće ulazne podatke:

```
Unesite broj elemenata niza (max 10): 8
Unesite A[0]: 1
Unesite A[1]: 2
Unesite A[2]: -2
Unesite A[3]: -1
Unesite A[4]: 3
Unesite A[5]: 1
```

```
Unesite A[6]: 2
Unesite A[7]: -1
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 1
A[1] = 2
A[2] = -2
A[3] = -1
A[4] = 3
A[5] = 1
A[6] = 2
A[7] = -1

abs(A[0]) == abs(A[3]) == 1
abs(A[0]) == abs(A[5]) == 1
abs(A[0]) == abs(A[7]) == 1
abs(A[1]) == abs(A[2]) == 2
abs(A[1]) == abs(A[6]) == 2
abs(A[2]) == abs(A[6]) == 2
abs(A[3]) == abs(A[5]) == 1
abs(A[3]) == abs(A[7]) == 1
abs(A[5]) == abs(A[7]) == 1

count = 9
```

2.12.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 10

int main() {
    int A[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite A[%d]: ", i);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    int j;
    int count = 0;
```

```
    printf("\n");
    for(i=0; i<n-1; i++) {
        for(j=i+1; j<n; j++) {
            if (abs(A[i]) == abs(A[j])) {
                printf("abs(A[%d]) == abs(A[%d]) == %d\n", i, j, abs(A[i]));
                count++;
            }
        }
    }

    printf("\ncount = %d\n", count);

    return 0;
}
```

2.13 Zadatak „Poslednje cifre jednake”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz `A` od maksimalno 10 pozitivnih celobrojnih elemenata. Učitati `n` elemenata a zatim:

- ispisati sadržaj celokupnog niza
- proveriti da li postoje parovi elemenata za koje važi da su im poslednje cifre jednake

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`

Za sledeće ulazne podatke:

```
Unesite broj elemenata niza (max 10): 8
Unesite A[0]: 1234
Unesite A[1]: 123
Unesite A[2]: 12
Unesite A[3]: 1
Unesite A[4]: 21
Unesite A[5]: 323
Unesite A[6]: 4321
Unesite A[7]: 6661999
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 1234
A[1] = 123
A[2] = 12
A[3] = 1
A[4] = 21
A[5] = 323
A[6] = 4321
A[7] = 6661999
```

```
last(A[1]) == last(A[5]) == 3
last(A[3]) == last(A[4]) == 1
last(A[3]) == last(A[6]) == 1
last(A[4]) == last(A[6]) == 1

count = 4
```

2.13.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 10

int last(int n) {
    return n % 10;
}

int main() {
    int A[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);

    int i;
    for(i=0; i<n; i++) {
        printf("Unesite A[%d]: ", i);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    int j;
    int count = 0;
    printf("\n");
    for(i=0; i<n-1; i++) {
        for(j=i+1; j<n; j++) {
            if (last(A[i]) == last(A[j])) {
                printf("last(A[%d]) == last(A[%d]) == %d\n", i, j, last(A[i]));
                count++;
            }
        }
    }

    printf("\ncount = %d\n", count);

    return 0;
}
```

2.14 Zadatak „Aritmetička sredina dela niza”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Dat je niz `A` od maksimalno 10 celobrojnih elemenata. Učitati `n` elemenata a zatim:

- ispisati sadržaj celokupnog niza
- odrediti i ispisati aritmetičku sredinu elemenata niza koji su na **parnim indeksima** (`AS`)

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`
- vrednost promenljive `AS` ispisati na 2 decimale

Za sledeće ulazne podatke:

```
Unesite broj elemenata niza (max 10): 5
Unesite A[0]: 4
Unesite A[1]: 2
Unesite A[2]: 6
Unesite A[3]: -7
Unesite A[4]: 1
```

očekivani izlaz je u sledećem formatu:

```
A[0] = 4
A[1] = 2
A[2] = 6
A[3] = -7
A[4] = 1

AS = 3.67
```

Korisne matematičke formule:

- aritmetička sredina niza $AS = \frac{1}{n} \sum x_i$

2.14.1 Primer rešenja

```c
#include <stdio.h>

#define MAX_SIZE 10

int main() {
    int A[MAX_SIZE];
    int n;

    do {
        printf("Unesite broj elemenata niza (max %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while(n<2 || n > MAX_SIZE);
```

```
    int i;
    for(i=0; i<n; i++) {
        printf("Unesite A[%d]: ", i);
        scanf("%d", &A[i]);
    }

    printf("\n");
    for(i=0; i<n; i++) {
        printf("A[%d] = %d\n", i, A[i]);
    }

    double AS_sum = 0;
    int AS_count = 0;
    for(i=0; i<n; i+=2) {
        AS_sum += A[i];
        AS_count++;
    }
    double AS = AS_sum / AS_count;
    printf("\nAS = %.2lf\n", AS);

    return 0;
}
```

2.15 Zadatak „Na kraju, može biti samo jedan!”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Učitati broj članova reda (n) a zatim odrediti približnu vrednost broja 1 primenom sledeće matematičke formule:

$$1 \approx \sum_{i=0}^{n-1} \frac{1}{(i+1)(i+2)} = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \cdots + \frac{1}{n(n+1)}$$

Ograničenja:

- sprovesti zaštitu unosa nad podatkom n
- sve realne promenljive trebaju biti tipa double
- ne koristiti promenljive tipa niz

Za sledeće ulazne podatke:

```
Unesite broj clanova reda: 5
```

očekivani izlaz je u sledećem formatu:

```
clan[i=0] = 0.500000
clan[i=1] = 0.166667
clan[i=2] = 0.083333
clan[i=3] = 0.050000
clan[i=4] = 0.033333

sum = 0.833333
```

2.15.1 Primer rešenja

```c
#include <stdio.h>

int main() {
    int n;

    do {
        printf("Unesite broj clanova reda: ");
        scanf("%d", &n);
    } while(n<1);

    int i, j;
    double brojilac, imenilac, clan;
    double sum = 0;
    for(i=0; i<n; i++) {
        brojilac = 1;

        imenilac = (i+1) * (i+2);

        clan = brojilac/imenilac;
        printf("clan[i=%d] = %lf\n", i, clan);

        sum += clan;
    }

    printf("\nsum = %lf\n", sum);

    return 0;
}
```

2.16 Zadatak „Zašto jednostavno kad može komplikovano”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Učitati broj članova reda (`n`) a zatim odrediti približnu vrednost broja $1/4$ primenom sledeće matematičke formule:

$$\frac{1}{4} \approx \sum_{i=1}^{n} \frac{1}{i(i+1)(i+2)} = \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \frac{1}{3 \cdot 4 \cdot 5} + \cdots + \frac{1}{n(n+1)(n+2)}$$

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`
- sve realne promenljive trebaju biti tipa `double`
- ne koristiti promenljive tipa niz

Za sledeće ulazne podatke:

```
Unesite broj clanova reda: 5
```

očekivani izlaz je u sledećem formatu:

```
clan[i=1] = 0.166667
clan[i=2] = 0.041667
clan[i=3] = 0.016667
clan[i=4] = 0.008333
clan[i=5] = 0.004762

sum = 0.238095
```

2.16.1 Primer rešenja

```c
#include <stdio.h>

int main() {
    int n;

    do {
        printf("Unesite broj clanova reda: ");
        scanf("%d", &n);
    } while(n<1);

    int i, j;
    double brojilac, imenilac, clan;
    double sum = 0;
    for(i=1; i<=n; i++) {
        brojilac = 1;

        imenilac = i * (i+1) * (i+2);

        clan = brojilac/imenilac;
        printf("clan[i=%d] = %lf\n", i, clan);

        sum += clan;
    }

    printf("\nsum = %lf\n", sum);

    return 0;
}
```

2.17 Zadatak „*e*-ksplodirajući rast”

Autor zadatka: Petar Marić <*petarmaric@uns.ac.rs*>

Učitati broj članova reda (n) a zatim odrediti približnu vrednost e primenom sledeće matematičke formule:

$$e \approx \sum_{i=0}^{n-1} \frac{1}{i!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{(n-1)!}$$

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`
- sve realne promenljive trebaju biti tipa `double`
- ne koristiti promenljive tipa niz

Za sledeće ulazne podatke:

```
Unesite broj clanova reda: 5
```

očekivani izlaz je u sledećem formatu:

```
clan[i=0] = 1.000000
clan[i=1] = 1.000000
clan[i=2] = 0.500000
clan[i=3] = 0.166667
clan[i=4] = 0.041667

e = 2.708333
```

2.17.1 Primer rešenja

```c
#include <stdio.h>
#include <math.h>

int main() {
    int n;

    do {
        printf("Unesite broj clanova reda: ");
        scanf("%d", &n);
    } while(n<1);

    int i, j;
    double brojilac, imenilac, clan;
    double sum = 0;
    for(i=0; i<n; i++) {
        brojilac = 1;

        double fakt = 1;
        for(j=2; j<=i; j++) {
            fakt *= j;
        }
        imenilac = fakt;

        clan = brojilac/imenilac;
        printf("clan[i=%d] = %lf\n", i, clan);

        sum += clan;
    }

    printf("\ne = %lf\n", sum);
```

```
    return 0;
}
```

2.18 Zadatak „Per factorial ad Euler”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Učitati broj članova reda (n) a zatim odrediti približnu vrednost e primenom sledeće matematičke formule:

$$\frac{1}{e} \approx \sum_{i=0}^{n-1} \frac{(-1)^i}{i!} = \frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + \frac{(-1)^{n-1}}{(n-1)!}$$

Ograničenja:

- sprovesti zaštitu unosa nad podatkom n
- sve realne promenljive trebaju biti tipa double
- ne koristiti promenljive tipa niz

Za sledeće ulazne podatke:

```
Unesite broj clanova reda: 5
```

očekivani izlaz je u sledećem formatu:

```
clan[i=0] =  1.000000
clan[i=1] = -1.000000
clan[i=2] =  0.500000
clan[i=3] = -0.166667
clan[i=4] =  0.041667

sum = 0.375000

e = 2.666667
```

2.18.1 Primer rešenja

```
#include <stdio.h>
#include <math.h>

int main() {
    int n;

    do {
        printf("Unesite broj clanova reda: ");
        scanf("%d", &n);
    } while(n<1);

    int i, j;
```

```
    double brojilac, imenilac, clan;
    double sum = 0;
    for(i=0; i<n; i++) {
        brojilac = pow(-1, i);

        double fakt = 1;
        for(j=2; j<=i; j++) {
            fakt *= j;
        }
        imenilac = fakt;

        clan = brojilac/imenilac;
        printf("clan[i=%d] = % lf\n", i, clan);

        sum += clan;
    }

    printf("\nsum = %lf\n", sum);

    double e = 1 / sum;
    printf("\ne = %lf\n", e);

    return 0;
}
```

2.19 Zadatak „Jedno parče *pi*-te"

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Učitati broj članova reda (`n`) a zatim odrediti približnu vrednost `pi` primenom sledeće matematičke formule:

$$\frac{\pi^2}{8} \approx \sum_{i=1}^{n} \frac{1}{(2i-1)^2} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \cdots + \frac{1}{(2n-1)^2}$$

Ograničenja:

- sprovesti zaštitu unosa nad podatkom `n`
- sve realne promenljive trebaju biti tipa `double`
- ne koristiti promenljive tipa niz

Za sledeće ulazne podatke:

```
Unesite broj clanova reda: 5
```

očekivani izlaz je u sledećem formatu:

```
clan[i=1] = 1.000000
clan[i=2] = 0.111111
clan[i=3] = 0.040000
clan[i=4] = 0.020408
```

```
clan[i=5] = 0.012346

sum = 1.183865

pi = 3.077486
```

2.19.1 Primer rešenja

```c
#include <stdio.h>
#include <math.h>

int main() {
    int n;

    do {
        printf("Unesite broj clanova reda: ");
        scanf("%d", &n);
    } while(n<1);

    int i, j;
    double brojilac, imenilac, clan;
    double sum = 0;
    for(i=1; i<=n; i++) {
        brojilac = 1;

        imenilac = pow(2*i - 1, 2);

        clan = brojilac/imenilac;
        printf("clan[i=%d] = %lf\n", i, clan);

        sum += clan;
    }

    printf("\nsum = %lf\n", sum);

    double pi = sqrt(sum * 8);
    printf("\npi = %lf\n", pi);

    return 0;
}
```

CHAPTER

3

PRIMERI ZADATAKA ZA T34

3.1 Zadatak „Korekcija živosti boja”

Autor zadatka: Petar Marić <*petarmaric@uns.ac.rs*>

Proizvođač je prilikom razvoja svog novog modela CCD foto-senzora napravio propust koji dovodi do „preslušavanja” između plavog i crvenog kanala slike, zapisane u 24-bit RGB kolor formatu. Pošto je ovaj problem uočen tek u kasnoj fazi razvoja, odlučeno je da će se ipak nastaviti proizvodnja novog modela senzora, uz PR kampanju *„It's not a bug; it's an undocumented feature!”*

Nezavisno od njih proizvođač profesionalnih fotoaparata sproveo je opsežno marketinško istraživanje o novom senzoru i njegovoj nadrealnoj živosti boja, ali ni uz silan uložen trud (i lobiranje) nažalost nije naišao na razumevanje udruženja modnih fotografa.

Stoga je proizvođač fotoaparata svojim inženjerima dao zadatak da softverski koriguju **plavi** kanal slike, primenom `XOR` bitwise operatora nad plavim i crvenim kanalom.

Primer poziva programa:

```
./fix-colors cat-weird.bmp cat-fixed.bmp
```

sa zadatim ulazom u datoteci `cat-weird.bmp`:

i očekivanim izlazom u datoteci `cat-fixed.bmp`:

Primer poziva programa:

```
./fix-colors dancers-weird.bmp dancers-fixed.bmp
```

sa zadatim ulazom u datoteci `dancers-weird.bmp`:

i očekivanim izlazom u datoteci `dancers-fixed.bmp`:

Izvori slika:

- https://www.flickr.com/photos/waferboard/5436402112 (CC BY 2.0)
- https://www.flickr.com/photos/dfataustralianaid/10691296013 (CC BY 2.0)

3.1.1 Primer rešenja

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define HEADER_SIZE 54
#define WIDTH_POS 18
```

```c
#define HEIGHT_POS 22

typedef struct pixel_st {
    unsigned char B;
    unsigned char G;
    unsigned char R;
} PIXEL;


FILE *safe_fopen(char *name, char *mode);
unsigned examine_image_header(FILE *in, unsigned char *header);
void read_image(FILE *in, PIXEL *pixels, unsigned image_size);
void write_image(FILE *out, unsigned char *header, PIXEL *pixels,
unsigned image_size);
void fix_colors(PIXEL *pixels, unsigned image_size);


int main(int num_args, char **args) {
    if(num_args != 3) {
        printf("USAGE: %s in.bmp out.bmp", args[0]);
        exit(1);
    }

    FILE *in = safe_fopen(args[1], "rb");
    FILE *out = safe_fopen(args[2], "wb");

    unsigned char header[HEADER_SIZE];
    unsigned image_size = examine_image_header(in, header);

    PIXEL pixels[image_size];
    read_image(in, pixels, image_size);

    fix_colors(pixels, image_size);

    write_image(out, header, pixels, image_size);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char *name, char *mode) {
    FILE *pf = fopen(name, mode);

    if(pf == NULL) {
        printf("File %s could not be opened!\n", name);
        exit(EXIT_FAILURE);
    }

    return pf;
}

unsigned examine_image_header(FILE *in, unsigned char *header) {
    fread(header, sizeof(unsigned char), HEADER_SIZE, in);

    unsigned width = *(unsigned *) &header[WIDTH_POS];
```

```
    unsigned height = *(unsigned *) &header[HEIGHT_POS];

    return width * height;
}

void read_image(FILE *in, PIXEL *pixels, unsigned image_size) {
    fread(pixels, sizeof(PIXEL), image_size, in);
}

void write_image(FILE *out, unsigned char *header, PIXEL *pixels,
unsigned image_size) {
    fwrite(header, sizeof(unsigned char), HEADER_SIZE, out);
    fwrite(pixels, sizeof(PIXEL), image_size, out);
}

void fix_colors(PIXEL *pixels, unsigned image_size) {
    unsigned i;
    unsigned char tmp;

    for(i=0; i<image_size; i++) {
        pixels[i].B ^= pixels[i].R;
    }
}
```

3.2 Zadatak „Analiza SMS cena”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteka učitati statički niz struktura, pri čemu se struktura `paket_st` sastoji od sledećih polja:

- Naziv provajdera (jedna reč, do 20 karaktera)
- Naziv paketa (jedna reč, do 20 karaktera)
- Mesečna pretplata (prirodan broj)
- Broj besplatnih poruka uključenih u pretplatu (prirodan broj)
- Cena po poruci nakon potrošenih besplatnih poruka (realan broj)

Na osnovu zadatog mesečnog broja poruka `mesecno_poruka` (prirodan broj) formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `cena_st` sastoji od sledećih polja:

- Ukupna cena paketa (zaokružena na 2 decimale, koristiti `"%7.2f"` format specifikator)
- Naziv provajdera (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `paket_st`)
- Naziv paketa (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `paket_st`)

Primer poziva:

```
./sms 237 paketi.txt cene.txt
```

sa `mesecno_poruka=237`, zadatim ulazom u datoteci `paketi.txt`:

```
mrs     belka50     360  50 3.89
mrs     belka100    660 100 3.89
mrs     belka150    900 150 3.89
teleLOL premesi100  600 100 3.97
teleLOL premesi220 1006 220 3.97
vim     smaragd100  590 100 3.90
vim     smaragd250  990 250 3.90
```

i očekivanim izlazom u datoteci `cene.txt`:

```
1087.43 mrs belka50
1192.93 mrs belka100
1238.43 mrs belka150
1143.89 teleLOL premesi100
1073.49 teleLOL premesi220
1124.30 vim smaragd100
 990.00 vim smaragd250
```

3.2.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NIZ 30
#define MAX_NAZIV 20+1

struct paket_st {
    char naziv_provajdera[MAX_NAZIV];
    char naziv_paketa[MAX_NAZIV];
    int mesecna_pretplata;
    int broj_besplatnih_poruka;
    double cena_po_poruci;
};

struct cena_st {
    double ukupna_cena;
    char naziv_provajdera[MAX_NAZIV];
    char naziv_paketa[MAX_NAZIV];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_pakete(FILE *in, struct paket_st paketi[], int *n);
double cena_paketa(struct paket_st paket, int mesecno_poruka);
void transform(struct paket_st paketi[], struct cena_st cene[], int n,
int mesecno_poruka);
void snimi_cene(FILE *out, struct cena_st cene[], int n);
```

```c
int main(int arg_num, char **args) {
    if (arg_num != 4) {
        printf("USAGE: %s MESECNO_PORUKA IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(1);
    }

    int mesecno_poruka = atoi(args[1]);
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct paket_st paketi[MAX_NIZ];
    struct cena_st cene[MAX_NIZ];
    int n;

    ucitaj_pakete(in, paketi, &n);
    transform(paketi, cene, n, mesecno_poruka);
    snimi_cene(out, cene, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_pakete(FILE *in, struct paket_st paketi[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %s %d %d %lf",
         paketi[*n].naziv_provajdera,
         paketi[*n].naziv_paketa,
        &paketi[*n].mesecna_pretplata,
        &paketi[*n].broj_besplatnih_poruka,
        &paketi[*n].cena_po_poruci
    ) != EOF) {
        (*n)++;
    }
}

double cena_paketa(struct paket_st paket, int mesecno_poruka) {
    double base = paket.mesecna_pretplata;

    int extra_sms = mesecno_poruka - paket.broj_besplatnih_poruka;
    if (extra_sms > 0) {
        base += extra_sms * paket.cena_po_poruci;
```

```
    }

    return base;
}

void transform(struct paket_st paketi[], struct cena_st cene[], int n,
int mesecno_poruka) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(cene[i].naziv_provajdera, paketi[i].naziv_provajdera);
        strcpy(cene[i].naziv_paketa, paketi[i].naziv_paketa);

        cene[i].ukupna_cena = cena_paketa(paketi[i], mesecno_poruka);
    }
}

void snimi_cene(FILE *out, struct cena_st cene[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%7.2f %s %s\n",
            cene[i].ukupna_cena,
            cene[i].naziv_provajdera,
            cene[i].naziv_paketa
        );
    }
}
```

3.3 Zadatak „Krvna slika”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteka učitati statički niz struktura, pri čemu se struktura `pacijent_st` sastoji od sledećih polja:

- Ime (jedna reč, do 20 karaktera)
- Prezime (jedna reč, do 20 karaktera)
- Holesterol (realan broj)
- Krvni pritisak (prirodan broj)
- Šećer u krvi (realan broj)

Na osnovu zadatog tipa analize `tip_analize` (`holesterol`, `pritisak` ili `secer`) i sledećih pravila:

Tip analize	Prihvatljiv opseg
holesterol	holesterol < 5.2
pritisak	90 ≤ pritisak < 120
secer	3.9 ≤ secer < 5.6

formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `analiza_st` sastoji od sledećih polja:

- Upozorenje (1 znak):
  - '+' ako je vrednost van prihvatljivog opsega
  - '-' ako je vrednost u okviru prihvatljivog opsega
- Ime (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `pacijent_st`)
- Prezime (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `pacijent_st`)

Primer poziva:

```
./proveri pritisak pacijenti.txt izvestaj.txt
```

sa `tip_analize=pritisak`, zadatim ulazom u datoteci `pacijenti.txt`:

```
Homer  Simpson 13.5 178 7.8
Skinny Pete     4.3  85 3.6
Johnny Bravo    4.8  90 5.4
Pizza  Lover    5.2 120 3.9
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
+ Homer Simpson
+ Skinny Pete
- Johnny Bravo
+ Pizza Lover
```

3.3.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NIZ 30
#define MAX_IME 20+1

struct pacijent_st {
    char ime[MAX_IME];
    char prezime[MAX_IME];
    double holesterol;
    unsigned pritisak;
    double secer;
};

struct analiza_st {
    char upozorenje;
    char ime[MAX_IME];
    char prezime[MAX_IME];
};
```

```c
typedef enum {HOLESTEROL, PRITISAK, SECER} TIP_ANALIZE;


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_pacijente(FILE *in, struct pacijent_st pacijenti[], int *n);
int analiza_ok(struct pacijent_st pacijent, TIP_ANALIZE tip_analize);
void transform(struct pacijent_st pacijenti[], struct analiza_st analize[],
int n, TIP_ANALIZE tip_analize);
void snimi_analize(FILE *out, struct analiza_st analize[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 4) {
        printf("USAGE: %s TIP_ANALIZE IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(1);
    }

    char *tip_analize_str = args[1];
    char *in_filename = args[2];
    char *out_filename = args[3];

    TIP_ANALIZE tip_analize;
    if(strcmp(tip_analize_str, "holesterol") == 0) {
        tip_analize = HOLESTEROL;
    } else if (strcmp(tip_analize_str, "pritisak") == 0) {
        tip_analize = PRITISAK;
    } else if (strcmp(tip_analize_str, "secer") == 0) {
        tip_analize = SECER;
    } else {
        printf("Nepoznat tip analize '%s'!", tip_analize_str);
        exit(42);
    }

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct pacijent_st pacijenti[MAX_NIZ];
    struct analiza_st analize[MAX_NIZ];
    int n;

    ucitaj_pacijente(in, pacijenti, &n);
    transform(pacijenti, analize, n, tip_analize);
    snimi_analize(out, analize, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!", filename);
        exit(error_code);
    }
    return fp;
```

```
}

void ucitaj_pacijente(FILE *in, struct pacijent_st pacijenti[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %s %lf %u %lf",
         pacijenti[*n].ime,
         pacijenti[*n].prezime,
        &pacijenti[*n].holesterol,
        &pacijenti[*n].pritisak,
        &pacijenti[*n].secer
    ) != EOF) {
        (*n)++;
    }
}

int analiza_ok(struct pacijent_st p, TIP_ANALIZE tip_analize) {
    switch (tip_analize) {
        case HOLESTEROL: return p.holesterol < 5.2;
        case PRITISAK:   return (p.pritisak >= 90) && (p.pritisak < 120);
        case SECER:      return (p.secer >= 3.9) && (p.secer < 5.6);
    }
}

void transform(struct pacijent_st pacijenti[], struct analiza_st analize[],
int n, TIP_ANALIZE tip_analize) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(analize[i].ime, pacijenti[i].ime);
        strcpy(analize[i].prezime, pacijenti[i].prezime);

        analize[i].upozorenje = analiza_ok(pacijenti[i], tip_analize) ? '-':'+';
    }
}

void snimi_analize(FILE *out, struct analiza_st analize[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%c %s %s\n",
            analize[i].upozorenje,
            analize[i].ime,
            analize[i].prezime
        );
    }
}
```

3.4 Zadatak „Indeks telesne mase (BMI)”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteka učitati statički niz struktura, pri čemu se struktura `pacijent_st` sastoji od sledećih polja:

- Ime (jedna reč, do 20 karaktera)

- Prezime (jedna reč, do 20 karaktera)
- Masa u kilogramima (prirodan broj)
- Visina u metrima (realan broj)

Kategorizacija indeksa telesne mase ($BMI = \frac{masa}{visina^2}$):

Indeks	Dijagnoza
BMI < 18.5	Pothranjenost
18.5 ≤ BMI < 25.0	Idealna tezina
25.0 ≤ BMI < 30.0	Prekomerna tezina
BMI ≥ 30.0	Gojaznost

Formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `analiza_st` sastoji od sledećih polja:

- Ime (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `pacijent_st`)
- Prezime (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `pacijent_st`)
- BMI (zaokružen na 2 decimale)
- Dijagnoza

Primer poziva:

```
./bmi pacijenti.txt izvestaj.txt
```

sa zadatim ulazom u datoteci `pacijenti.txt`:

```
Homer  Simpson 150 1.83
Skinny Pete     61 1.90
Johnny Bravo    75 1.97
Pizza  Lover    81 1.80
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
Homer Simpson 44.79 Gojaznost
Skinny Pete 16.90 Pothranjenost
Johnny Bravo 19.33 Idealna tezina
Pizza Lover 25.00 Prekomerna tezina
```

3.4.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NIZ 30
#define MAX_IME 20+1
```

```c
#define MAX_DIJAGNOZA 20+1

struct pacijent_st {
    char ime[MAX_IME];
    char prezime[MAX_IME];
    unsigned masa;
    double visina;
};

struct analiza_st {
    char ime[MAX_IME];
    char prezime[MAX_IME];
    double bmi;
    char dijagnoza[MAX_DIJAGNOZA];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_pacijente(FILE *in, struct pacijent_st pacijenti[], int *n);
double odredi_bmi(struct pacijent_st pacijent);
void transform(struct pacijent_st pacijenti[], struct analiza_st analize[],
int n);
void snimi_analize(FILE *out, struct analiza_st analize[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 3) {
        printf("USAGE: %s IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(1);
    }

    char *in_filename = args[1];
    char *out_filename = args[2];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct pacijent_st pacijenti[MAX_NIZ];
    struct analiza_st analize[MAX_NIZ];
    int n;

    ucitaj_pacijente(in, pacijenti, &n);
    transform(pacijenti, analize, n);
    snimi_analize(out, analize, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!", filename);
        exit(error_code);
    }
```

```
    return fp;
}

void ucitaj_pacijente(FILE *in, struct pacijent_st pacijenti[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %s %u %lf",
         pacijenti[*n].ime,
         pacijenti[*n].prezime,
        &pacijenti[*n].masa,
        &pacijenti[*n].visina
    ) != EOF) {
        (*n)++;
    }
}

double odredi_bmi(struct pacijent_st pacijent) {
    return pacijent.masa / (pacijent.visina * pacijent.visina);
}

void transform(struct pacijent_st pacijenti[], struct analiza_st analize[],
int n) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(analize[i].ime, pacijenti[i].ime);
        strcpy(analize[i].prezime, pacijenti[i].prezime);

        double bmi = odredi_bmi(pacijenti[i]);
        analize[i].bmi = bmi;

        if (bmi < 18.5) {
            strcpy(analize[i].dijagnoza, "Pothranjenost");
        } else if (bmi < 25.0) {
            strcpy(analize[i].dijagnoza, "Idealna tezina");
        } else if (bmi < 30.0) {
            strcpy(analize[i].dijagnoza, "Prekomerna tezina");
        } else {
            strcpy(analize[i].dijagnoza, "Gojaznost");
        }
    }
}

void snimi_analize(FILE *out, struct analiza_st analize[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%s %s %.2f %s\n",
            analize[i].ime,
            analize[i].prezime,
            analize[i].bmi,
            analize[i].dijagnoza
        );
    }
}
```

3.5 Zadatak „Hipermarket”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `artikal_st` sastoji od sledećih polja:

- cena (pozitivan realan broj)
- broj prodatih artikala (prirodan broj)
- trgovačka marža u procentima (pozitivan realan broj)
- naziv (jedna reč, do 20 karaktera)

Formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `zarada_st` sastoji od sledećih polja:

- ostvarena zarada (zaokružena na 2 decimale, koristiti `"%7.2f"` format specifikator)
- naziv (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `artikal_st`)

Primer poziva:

```
./hipermarket artikli.txt zarada.txt
```

sa zadatim ulazom u datoteci `artikli.txt`:

```
  40.00  20  5.0 Politika
  50.00 800 10.5 Hleb
2219.99   2 30.0 Parmezan
 118.36  76 12.3 Mleko
```

i očekivanim izlazom u datoteci `zarada.txt`:

```
  40.00 Politika
4200.00 Hleb
1331.99 Parmezan
1106.43 Mleko
```

3.5.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV 20+1
#define MAX_NIZ 30

struct artikal_st {
    double cena;
    unsigned broj_prodatih;
    double trgovacka_marza;
```

```
    char naziv[MAX_NAZIV];
};

struct zarada_st {
    double zarada;
    char naziv[MAX_NAZIV];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_artikle(FILE *in, struct artikal_st artikli[], int *n);
double odredi_zaradu(struct artikal_st artikal);
void transform(struct artikal_st artikli[], struct zarada_st zarade[], int n);
void snimi_zarade(FILE *out, struct zarada_st zarade[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 3) {
        printf("USAGE: %s IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(42);
    }

    char *in_filename = args[1];
    char *out_filename = args[2];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct artikal_st artikli[MAX_NIZ];
    struct zarada_st zarade[MAX_NIZ];
    int n;

    ucitaj_artikle(in, artikli, &n);
    transform(artikli, zarade, n);
    snimi_zarade(out, zarade, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_artikle(FILE *in, struct artikal_st artikli[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%lf %u %lf %s",
        &artikli[*n].cena,
        &artikli[*n].broj_prodatih,
```

```
        &artikli[*n].trgovacka_marza,
         artikli[*n].naziv
    ) != EOF) {
        (*n)++;
    }
}

double odredi_zaradu(struct artikal_st artikal) {
    return artikal.cena * artikal.broj_prodatih * artikal.trgovacka_marza / 100;
}

void transform(struct artikal_st artikli[], struct zarada_st zarade[], int n) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(zarade[i].naziv, artikli[i].naziv);

        zarade[i].zarada = odredi_zaradu(artikli[i]);
    }
}

void snimi_zarade(FILE *out, struct zarada_st zarade[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%7.2f %s\n",
            zarade[i].zarada,
            zarade[i].naziv
        );
    }
}
```

3.6 Zadatak „Poskupljenja goriva (ni)su relativna”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `gorivo_st` sastoji od sledećih polja:

- skraćena oznaka grada (jedna reč, tačno 2 karaktera)
- tip goriva (jedna reč, do 10 karaktera)
- cena goriva (pozitivan realan broj)

Na osnovu fiksnog iznosa povećanja akcize na sva goriva `akciza` (pozitivan realan broj) formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `analiza_st` sastoji od sledećih polja:

- nova cena goriva (zaokružena na 2 decimale, koristiti „%6.2f” format specifikator)
- procenat uvećanja cene goriva (zaokružen na 2 decimale, koristiti „%5.2f” format specifikator)

- skraćena oznaka grada (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `gorivo_st`)
- tip goriva (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `gorivo_st`)

Primer poziva:

```
./ekologija 7.32 pre.txt posle.txt
```

sa `akciza=7.32` i zadatim ulazom u datoteci `pre.txt`:

```
NS gas     71.90
NS benzin 155.20
NS dizel  165.15
NI gas     69.30
NI benzin 155.30
NI dizel  165.20
BG gas     70.20
BG benzin 155.50
BG dizel  165.40
```

i očekivanim izlazom u datoteci `posle.txt`:

```
 79.22 10.18 NS gas
162.52  4.72 NS benzin
172.47  4.43 NS dizel
 76.62 10.56 NI gas
162.62  4.71 NI benzin
172.52  4.43 NI dizel
 77.52 10.43 BG gas
162.82  4.71 BG benzin
172.72  4.43 BG dizel
```

3.6.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_GRAD 2+1
#define MAX_GORIVO 10+1
#define MAX_NIZ 30

struct gorivo_st {
    char grad[MAX_GRAD];
    char tip_goriva[MAX_GORIVO];
    double cena;
};

struct analiza_st {
    double nova_cena;
    double procenat_uvecanja_cene;
    char grad[MAX_GRAD];
    char tip_goriva[MAX_GORIVO];
```

```
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_goriva(FILE *in, struct gorivo_st goriva[], int *n);
double odredi_novu_cenu(struct gorivo_st gorivo, double akciza);
double odredi_procenat_uvecanja_cene(double stara_cena, double nova_cena);
void transform(struct gorivo_st goriva[], struct analiza_st analize[], int n,
double akciza);
void snimi_analize(FILE *out, struct analiza_st analize[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 4) {
        printf("USAGE: %s AKCIZA IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(42);
    }

    double akciza = atof(args[1]);
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct gorivo_st goriva[MAX_NIZ];
    struct analiza_st analize[MAX_NIZ];
    int n;

    ucitaj_goriva(in, goriva, &n);
    transform(goriva, analize, n, akciza);
    snimi_analize(out, analize, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_goriva(FILE *in, struct gorivo_st goriva[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %s %lf",
         goriva[*n].grad,
         goriva[*n].tip_goriva,
        &goriva[*n].cena
    ) != EOF) {
        (*n)++;
```

```
    }
}

double odredi_novu_cenu(struct gorivo_st gorivo, double akciza) {
    return gorivo.cena + akciza;
}
double odredi_procenat_uvecanja_cene(double stara_cena, double nova_cena) {
    return (nova_cena / stara_cena - 1) * 100;
}

void transform(struct gorivo_st goriva[], struct analiza_st analize[], int n,
double akciza) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(analize[i].grad, goriva[i].grad);
        strcpy(analize[i].tip_goriva, goriva[i].tip_goriva);

        analize[i].nova_cena = odredi_novu_cenu(goriva[i], akciza);
        analize[i].procenat_uvecanja_cene = odredi_procenat_uvecanja_cene(
            goriva[i].cena, analize[i].nova_cena
        );
    }
}

void snimi_analize(FILE *out, struct analiza_st analize[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%6.2f %5.2f %s %s\n",
            analize[i].nova_cena,
            analize[i].procenat_uvecanja_cene,
            analize[i].grad,
            analize[i].tip_goriva
        );
    }
}
```

3.7 Zadatak „Novogodišnja žurka”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `zelja_st` sastoji od sledećih polja:

- broj gostiju (prirodan broj)
- naziv željene pizze (jedna reč, do 20 karaktera)

Na osnovu prosečnog apetita gostiju u parčadima pizze `apetit` (pozitivan realan broj) formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `porudzbina_st` sastoji od sledećih polja:

- broj potrebnih parčadi (zaokružen na 1 decimalu, koristiti `"%5.1f"` format specifikator)

- broj potrebnih *celih* pizza (koristiti `"%2u"` format specifikator)
- naziv željene pizze (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `zelja_st`)

Smatrati da se jedna cela pizza seče na 8 jednakih parčadi.

Primer poziva:

```
./zurka 2.5 zelje.txt nabavka.txt
```

sa `apetit=2.5` i zadatim ulazom u datoteci `zelje.txt`:

```
40 Capricciosa
27 Madjarica
11 Margharita
 3 Vegetale
16 Prosciutto
```

i očekivanim izlazom u datoteci `nabavka.txt`:

```
100.0 13 Capricciosa
 67.5  9 Madjarica
 27.5  4 Margharita
  7.5  1 Vegetale
 40.0  5 Prosciutto
```

3.7.1 Primer rešenja

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BROJ_PARCADI_PO_PIZZI 8
#define MAX_NAZIV 20+1
#define MAX_NIZ 30

struct zelja_st {
    unsigned broj_gostiju;
    char pizza[MAX_NAZIV];
};

struct porudzbina_st {
    double broj_parcadi;
    unsigned broj_celih_pizza;
    char pizza[MAX_NAZIV];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_zelje(FILE *in, struct zelja_st zelje[], int *n);
double odredi_broj_parcadi(struct zelja_st zelja, double apetit);
unsigned odredi_broj_celih_pizza(double broj_parcadi);
void transform(struct zelja_st zelje[], struct porudzbina_st porudzbine[],
```

```
int n, double apetit);
void snimi_porudzbine(FILE *out, struct porudzbina_st porudzbine[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 4) {
        printf("USAGE: %s APETIT IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(42);
    }

    double apetit = atof(args[1]);
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct zelja_st zelje[MAX_NIZ];
    struct porudzbina_st porudzbine[MAX_NIZ];
    int n;

    ucitaj_zelje(in, zelje, &n);
    transform(zelje, porudzbine, n, apetit);
    snimi_porudzbine(out, porudzbine, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_zelje(FILE *in, struct zelja_st zelje[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%u %s",
        &zelje[*n].broj_gostiju,
         zelje[*n].pizza
    ) != EOF) {
        (*n)++;
    }
}

double odredi_broj_parcadi(struct zelja_st zelja, double apetit) {
    return zelja.broj_gostiju * apetit;
}

unsigned odredi_broj_celih_pizza(double broj_parcadi) {
    return (unsigned) ceil(broj_parcadi / BROJ_PARCADI_PO_PIZZI);
```

```
}

void transform(struct zelja_st zelje[], struct porudzbina_st porudzbine[],
int n, double apetit) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(porudzbine[i].pizza, zelje[i].pizza);

        porudzbine[i].broj_parcadi = odredi_broj_parcadi(zelje[i], apetit);
        porudzbine[i].broj_celih_pizza = odredi_broj_celih_pizza(
            porudzbine[i].broj_parcadi
        );
    }
}

void snimi_porudzbine(FILE *out, struct porudzbina_st porudzbine[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%5.1f %2u %s\n",
            porudzbine[i].broj_parcadi,
            porudzbine[i].broj_celih_pizza,
            porudzbine[i].pizza
        );
    }
}
```

3.8 Zadatak „Cena struje u regionu”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `paket_st` sastoji od sledećih polja:

- naziv zemlje (jedna reč, do 20 karaktera)
- naziv operatera (jedna reč, do 20 karaktera)
- mesečna pretplata (prirodan broj)
- broj KWh uključenih u pretplatu (prirodan broj)
- cena po KWh nakon potrošenih besplatnih KWh (pozitivan realan broj)

Na osnovu zadatog mesečnog broja KWh `potroseno_kwh` (prirodan broj) formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `cena_st` sastoji od sledećih polja:

- ukupna cena (zaokružena na 2 decimale, koristiti `"%7.2f"` format specifikator)
- naziv zemlje (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `paket_st`)

- naziv operatera (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `paket_st`)

Primer poziva:

```
./potrosnja 90 paketi.txt cene.txt
```

sa `potroseno_kwh=90` i zadatim ulazom u datoteci `paketi.txt`:

```
Srbija     EPS             500  50  3.92
Srbija     TajkunDOO       100   2  9.56
Madjarska  HuElectro       495 100 12.31
Slovenija  Elektroprivreda  50   0 15.99
Slovenija  AlpEnergy         0   0 16.99
```

i očekivanim izlazom u datoteci `cene.txt`:

```
 656.80 Srbija EPS
 941.28 Srbija TajkunDOO
 495.00 Madjarska HuElectro
1489.10 Slovenija Elektroprivreda
1529.10 Slovenija AlpEnergy
```

3.8.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV 20+1
#define MAX_NIZ 30

struct paket_st {
    char zemlja[MAX_NAZIV];
    char operater[MAX_NAZIV];
    unsigned mesecna_pretplata;
    unsigned broj_besplatnih_kwh;
    double cena_po_kwh;
};

struct cena_st {
    double ukupna_cena;
    char zemlja[MAX_NAZIV];
    char operater[MAX_NAZIV];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_pakete(FILE *in, struct paket_st paketi[], int *n);
double odredi_ukupnu_cenu(struct paket_st paket, int potroseno_kwh);
void transform(struct paket_st paketi[], struct cena_st cene[], int n,
int potroseno_kwh);
void snimi_cene(FILE *out, struct cena_st cene[], int n);
```

```c
int main(int arg_num, char **args) {
    if (arg_num != 4) {
        printf("USAGE: %s POTROSENO_KWH IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(42);
    }

    int potroseno_kwh = atoi(args[1]);
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct paket_st paketi[MAX_NIZ];
    struct cena_st cene[MAX_NIZ];
    int n;

    ucitaj_pakete(in, paketi, &n);
    transform(paketi, cene, n, potroseno_kwh);
    snimi_cene(out, cene, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_pakete(FILE *in, struct paket_st paketi[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %s %u %u %lf",
         paketi[*n].zemlja,
         paketi[*n].operater,
        &paketi[*n].mesecna_pretplata,
        &paketi[*n].broj_besplatnih_kwh,
        &paketi[*n].cena_po_kwh
    ) != EOF) {
        (*n)++;
    }
}

double odredi_ukupnu_cenu(struct paket_st paket, int potroseno_kwh) {
    double base = paket.mesecna_pretplata;

    int extra_kwh = potroseno_kwh - paket.broj_besplatnih_kwh;
    if (extra_kwh > 0) {
        base += extra_kwh * paket.cena_po_kwh;
    }
```

```
    return base;
}

void transform(struct paket_st paketi[], struct cena_st cene[], int n,
int potroseno_kwh) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(cene[i].zemlja, paketi[i].zemlja);
        strcpy(cene[i].operater, paketi[i].operater);

        cene[i].ukupna_cena = odredi_ukupnu_cenu(paketi[i], potroseno_kwh);
    }
}

void snimi_cene(FILE *out, struct cena_st cene[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%7.2f %s %s\n",
            cene[i].ukupna_cena,
            cene[i].zemlja,
            cene[i].operater
        );
    }
}
```

3.9 Zadatak „Leti brzo, leti skupo”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

> Ko leti - vredi,
> ko vredi - leti,
> ko ne leti - ne vredi.
>
> —Otto Grunf

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `prevoznik_st` sastoji od sledećih polja:

- naziv operatera (jedna reč, do 20 karaktera)
- cena karte (pozitivan realan broj)
- maksimalna masa besplatnog prtljaga (pozitivan realan broj)
- doplata za prekomerni prtljag po kilogramu (pozitivan realan broj)

Na osnovu zadate mase prtljaga `masa` (pozitivan realan broj) formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `cena_st` sastoji od sledećih polja:

- ukupna cena leta (zaokružena na 2 decimale, koristiti `"%7.2f"` format specifikator)

- naziv operatera (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `prevoznik_st`)

Primer poziva:

```
./letovi 23.5 avioprevoznici.txt cene.txt
```

sa `masa=23.5` i zadatim ulazom u datoteci `avioprevoznici.txt`:

```
JAT       150.0 2.5 1.5
Wizz       52.3 0.0 7.0
Lufthansa 210.0 7.0 2.5
Delta     180.0 6.5 1.0
```

i očekivanim izlazom u datoteci `cene.txt`:

```
181.50 JAT
216.80 Wizz
251.25 Lufthansa
197.00 Delta
```

3.9.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV 20+1
#define MAX_NIZ 30

struct prevoznik_st {
    char operater[MAX_NAZIV];
    double cena_karte;
    double max_masa_besplatnog_prtljaga;
    double cena_po_kg;
};

struct cena_st {
    double ukupna_cena;
    char operater[MAX_NAZIV];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_prevoznike(FILE *in, struct prevoznik_st prevoznici[], int *n);
double odredi_ukupnu_cenu(struct prevoznik_st prevoznik, double masa);
void transform(struct prevoznik_st prevoznici[], struct cena_st cene[], int n,
double masa);
void snimi_cene(FILE *out, struct cena_st cene[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 4) {
        printf("USAGE: %s MASA IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(42);
```

```
    }

    double masa = atof(args[1]);
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct prevoznik_st prevoznici[MAX_NIZ];
    struct cena_st cene[MAX_NIZ];
    int n;

    ucitaj_prevoznike(in, prevoznici, &n);
    transform(prevoznici, cene, n, masa);
    snimi_cene(out, cene, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_prevoznike(FILE *in, struct prevoznik_st prevoznici[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %lf %lf %lf",
         prevoznici[*n].operater,
        &prevoznici[*n].cena_karte,
        &prevoznici[*n].max_masa_besplatnog_prtljaga,
        &prevoznici[*n].cena_po_kg
    ) != EOF) {
        (*n)++;
    }
}

double odredi_ukupnu_cenu(struct prevoznik_st prevoznik, double masa) {
    double base = prevoznik.cena_karte;

    double extra_masa = masa - prevoznik.max_masa_besplatnog_prtljaga;
    if (extra_masa > 0) {
        base += extra_masa * prevoznik.cena_po_kg;
    }

    return base;
}

void transform(struct prevoznik_st prevoznici[], struct cena_st cene[], int n,
```

```
double masa) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(cene[i].operater, prevoznici[i].operater);

        cene[i].ukupna_cena = odredi_ukupnu_cenu(prevoznici[i], masa);
    }
}

void snimi_cene(FILE *out, struct cena_st cene[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%7.2f %s\n",
            cene[i].ukupna_cena,
            cene[i].operater
        );
    }
}
```

3.10 Zadatak „Neizbežna posledica života”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

> U životu su izvesne samo dve stvari – smrt i porez.
>
> —Bendžamin Frenklin (1706 - 1790)

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `poreska_stopa_st` sastoji od sledećih polja:

- skraćena oznaka grada (jedna reč, tačno 2 karaktera)
- osnovni porez (prirodan broj)
- dodatni porez po kvadratu nekretnine (pozitivan realan broj)
- procenat poreskog popusta po članu domaćinstva (pozitivan realan broj)

Na osnovu površine nekretnine `povrsina` (pozitivan realan broj) i broja članova domaćinstva `clanova` (prirodan broj) formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `porez_st` sastoji od sledećih polja:

- skraćena oznaka grada (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `poreska_stopa_st`)
- ukupan porez bez popusta (zaokružen na 0 decimala, koristiti `"%5.0f"` format specifikator)
- ukupan porez sa popustom (zaokružen na 0 decimala, koristiti `"%5.0f"` format specifikator)

Primer poziva:

```
./porez 74.1 3 poreske_stope.txt porezi.txt
```

sa `povrsina=74.1` i `clanova=3`, zadatim ulazom u datoteci `poreske_stope.txt`:

```
NS 1712 67.1 1.1
NI  800 42.3 1.7
BG 3270 98.6 1.5
SU 1000 47.5 1.3
```

i očekivanim izlazom u datoteci `porezi.txt`:

```
NS  6684  6464
NI  3934  3734
BG 10576 10100
SU  4520  4343
```

3.10.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NIZ 30
#define MAX_NAZIV_GRADA 2+1

struct poreska_stopa_st {
    char grad[MAX_NAZIV_GRADA];
    unsigned osnovni_porez;
    double porez_po_kvadratu;
    double popust_po_clanu;
};

struct porez_st {
    char grad[MAX_NAZIV_GRADA];
    double ukupan_porez;
    double ukupan_porez_sa_popustom;
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_poreske_stope(FILE *in, struct poreska_stopa_st stope[], int *n);
double odredi_ukupan_porez(struct poreska_stopa_st stopa, double povrsina);
double odredi_ukupan_porez_sa_popustom(double ukupan_porez, int clanova,
double popust_po_clanu);
void transform(struct poreska_stopa_st stope[], struct porez_st porezi[], int n,
double povrsina, int clanova);
void snimi_poreze(FILE *out, struct porez_st porezi[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 5) {
        printf("USAGE: %s POVRSINA CLANOVA IN_FILE OUT_FILE\n", args[0]);
        exit(1);
    }
```

```
    double povrsina = atof(args[1]);
    int clanova = atoi(args[2]);
    char *in_filename = args[3];
    char *out_filename = args[4];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct poreska_stopa_st stope[MAX_NIZ];
    struct porez_st porezi[MAX_NIZ];
    int n;

    ucitaj_poreske_stope(in, stope, &n);
    transform(stope, porezi, n, povrsina, clanova);
    snimi_poreze(out, porezi, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_poreske_stope(FILE *in, struct poreska_stopa_st stope[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %u %lf %lf",
         stope[*n].grad,
        &stope[*n].osnovni_porez,
        &stope[*n].porez_po_kvadratu,
        &stope[*n].popust_po_clanu
    ) != EOF) {
        (*n)++;
    }
}

double odredi_ukupan_porez(struct poreska_stopa_st stopa, double povrsina) {
    return stopa.osnovni_porez + stopa.porez_po_kvadratu * povrsina;
}

double odredi_ukupan_porez_sa_popustom(double ukupan_porez, int clanova,
double popust_po_clanu) {
    double ukupan_popust = clanova * popust_po_clanu / 100;
    return ukupan_porez * (1 - ukupan_popust);
}

void transform(struct poreska_stopa_st stope[], struct porez_st porezi[], int n,
double povrsina, int clanova) {
    int i;
```

```
    for(i=0; i<n; i++) {
        strcpy(porezi[i].grad, stope[i].grad);

        porezi[i].ukupan_porez = odredi_ukupan_porez(stope[i], povrsina);
        porezi[i].ukupan_porez_sa_popustom = odredi_ukupan_porez_sa_popustom(
            porezi[i].ukupan_porez, clanova, stope[i].popust_po_clanu
        );
    }
}

void snimi_poreze(FILE *out, struct porez_st porezi[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%s %5.0f %5.0f\n",
            porezi[i].grad,
            porezi[i].ukupan_porez,
            porezi[i].ukupan_porez_sa_popustom
        );
    }
}
```

3.11 Zadatak „H-faktor”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `serija_st` sastoji od sledećih polja:

- naziv (jedna reč, do 30 karaktera)
- IMDB ocena (pozitivan realan broj)
- maksimalna zabeležena gledanost (pozitivan realan broj)
- maksimalan broj meseci između dve sezone (prirodan broj)

Formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `hype_st` sastoji od sledećih polja:

- hype faktor (zaokružen na 2 decimale, koristiti `"%6.2f"` format specifikator)
- naziv (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `serija_st`)

Korisne matematičke formule:

```
hype_faktor = imdb_ocena * max_vreme_izmedju_sezona / max_gledanost
```

Primer poziva:

```
./hype_train serije.txt hype.txt
```

sa zadatim ulazom u datoteci `serije.txt`:

```
SlickAndSmorty        9.3   2.86  18
EastWorld             8.8   2.24  16
GameOfPhones          9.5  12.07  20
VentureSisters        8.5   1.53  31
EvangerionNoYakusoku  8.0   2.16  59
```

i očekivanim izlazom u datoteci hype.txt:

```
 58.53 SlickAndSmorty
 62.86 EastWorld
 15.74 GameOfPhones
172.22 VentureSisters
218.52 EvangerionNoYakusoku
```

3.11.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NIZ 30
#define MAX_NAZIV 30+1

struct serija_st {
    char naziv[MAX_NAZIV];
    double imdb_ocena;
    double max_gledanost;
    unsigned max_vreme_izmedju_sezona;
};

struct hype_st {
    double hype_faktor;
    char naziv[MAX_NAZIV];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_serije(FILE *in, struct serija_st serije[], int *n);
void transform(struct serija_st serije[], struct hype_st hypetrain[], int n);
void snimi_hypetrain(FILE *out, struct hype_st hypetrain[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 3) {
        printf("USAGE: %s IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(42);
    }

    char *in_filename = args[1];
    char *out_filename = args[2];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct serija_st serije[MAX_NIZ];
```

```
    struct hype_st hypetrain[MAX_NIZ];
    int n;

    ucitaj_serije(in, serije, &n);
    transform(serije, hypetrain, n);
    snimi_hypetrain(out, hypetrain, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!", filename);
        exit(error_code);
    }
    return fp;
}

void ucitaj_serije(FILE *in, struct serija_st serije[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %lf %lf %u",
         serije[*n].naziv,
        &serije[*n].imdb_ocena,
        &serije[*n].max_gledanost,
        &serije[*n].max_vreme_izmedju_sezona
    ) != EOF) {
        (*n)++;
    }
}

double feel_the_hype(struct serija_st s) {
    return s.imdb_ocena * s.max_vreme_izmedju_sezona / s.max_gledanost;
}

void transform(struct serija_st serije[], struct hype_st hypetrain[], int n) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(hypetrain[i].naziv, serije[i].naziv);

        hypetrain[i].hype_faktor = feel_the_hype(serije[i]);
    }
}

void snimi_hypetrain(FILE *out, struct hype_st hypetrain[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%6.2lf %s\n",
            hypetrain[i].hype_faktor,
            hypetrain[i].naziv
        );
    }
```

```
}
```

3.12 Zadatak „15 miliona u crnom ogledalu”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati statički niz struktura, pri čemu se struktura `takmicar_st` sastoji od sledećih polja:

- ime (jedna reč, do 16 karaktera)
- prezime (jedna reč, do 16 karaktera)
- ocena prvog člana stručnog žirija (prirodan broj)
- ocena drugog člana stručnog žirija (prirodan broj)
- ocena trećeg člana stručnog žirija (prirodan broj)

Formirati novi statički niz struktura i upisati ga u zadatu izlaznu datoteku, pri čemu se struktura `ocena_st` sastoji od sledećih polja:

- prosečna ocena stručnog žirija (zaokružena na 2 decimale, koristiti `"%4.2f"` format specifikator)
- ime (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `takmicar_st`)
- prezime (koristiti funkciju `strcpy` prilikom kopiranja iz strukture `takmicar_st`)

Primer poziva:

```
./hotshot takmicari.txt ocene.txt
```

sa zadatim ulazom u datoteci `takmicari.txt`:

```
Bingham  Madsen  5  5  5
Abi      Khan    5  2  1
Glee     Hayes   1  0  0
```

i očekivanim izlazom u datoteci `ocene.txt`:

```
5.00 Bingham Madsen
2.67 Abi Khan
0.33 Glee Hayes
```

3.12.1 Primer rešenja

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
#define MAX_NIZ 30
#define MAX_IME 16+1

struct takmicar_st {
    char ime[MAX_IME];
    char prezime[MAX_IME];
    unsigned ocena_1;
    unsigned ocena_2;
    unsigned ocena_3;
};

struct ocena_st {
    double prosecna_ocena;
    char ime[MAX_IME];
    char prezime[MAX_IME];
};


FILE *safe_fopen(char filename[], char mode[], int error_code);
void ucitaj_takmicare(FILE *in, struct takmicar_st takmicari[], int *n);
void transform(struct takmicar_st takmicari[], struct ocena_st ocene[], int n);
void snimi_ocene(FILE *out, struct ocena_st ocene[], int n);


int main(int arg_num, char **args) {
    if (arg_num != 3) {
        printf("USAGE: %s IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(42);
    }

    char *in_filename = args[1];
    char *out_filename = args[2];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    struct takmicar_st takmicari[MAX_NIZ];
    struct ocena_st ocene[MAX_NIZ];
    int n;

    ucitaj_takmicare(in, takmicari, &n);
    transform(takmicari, ocene, n);
    snimi_ocene(out, ocene, n);

    fclose(in);
    fclose(out);

    return 0;
}

FILE *safe_fopen(char filename[], char mode[], int error_code) {
    FILE *fp = fopen(filename, mode);
    if(fp == NULL) {
        printf("Can't open '%s'!", filename);
        exit(error_code);
    }
    return fp;
```

```
}

void ucitaj_takmicare(FILE *in, struct takmicar_st takmicari[], int *n) {
    *n = 0;
    while(fscanf(
        in, "%s %s %u %u %u",
         takmicari[*n].ime,
         takmicari[*n].prezime,
        &takmicari[*n].ocena_1,
        &takmicari[*n].ocena_2,
        &takmicari[*n].ocena_3
    ) != EOF) {
        (*n)++;
    }
}

double odredi_prosecnu_ocenu(struct takmicar_st takmicar) {
    return (takmicar.ocena_1 + takmicar.ocena_2 + takmicar.ocena_3) / 3.;
}

void transform(struct takmicar_st takmicari[], struct ocena_st ocene[], int n) {
    int i;
    for(i=0; i<n; i++) {
        strcpy(ocene[i].ime, takmicari[i].ime);
        strcpy(ocene[i].prezime, takmicari[i].prezime);

        ocene[i].prosecna_ocena = odredi_prosecnu_ocenu(takmicari[i]);
    }
}

void snimi_ocene(FILE *out, struct ocena_st ocene[], int n) {
    int i;
    for(i=0; i<n; i++) {
        fprintf(
            out, "%4.2lf %s %s\n",
            ocene[i].prosecna_ocena,
            ocene[i].ime,
            ocene[i].prezime
        );
    }
}
```

CHAPTER

4

PRIMERI ZADATAKA ZA PREDMETNI PROJEKAT

4.1 Zadatak „50 nijansi istine”

Autor zadatka: Petar Marić <*petarmaric@uns.ac.rs*>

Stanovnici jednog grada su se pobunili zbog loših uslova života u divljim naseljima:

Figure 1: Izvor: https://en.wikipedia.org/wiki/File:Jakarta_slumhome_2.jpg (CC BY 2.0)

Gradska vlast je stoga zamolila nezavisne lokalne novinare da sprovedu opsežno i objektivno istraživanje, koje će *zasigurno* pokazati da je situacija na terenu fina i ružičasta. Nažalost, novinari su svoj zadatak doslovno shvatili:

Figure 2: `truth_factor = 0.4`

Gradskim ocima se srećom ipak svideo ovakav način izveštavanja, samo su smatrali da bi valjalo intenzivnije istaći realnost situacije na terenu:

Figure 3: `truth_factor = 0.7`

Nažalost, nezahvalan narod se još uvek bunio te je stoga upotrebljen još viši stepen algoritma za *optimizaciju percepcije* ™:

Figure 4: `truth_factor = 1.0`

Zahvaljujući ovako naprednoj tehnologiji objektivnog izveštavanja gradska vlast je najzad bila u mogućnosti da na jednostavan način široj javnosti pokaže da se problem na kojem su izvesni remetilački faktori tako uporno insistirali ne može primetiti ni u naznakama.

Novinari su pritom koristili sledeći algoritam za *optimizaciju percepcije* ™:

```
true_R = R * (1 - truth_factor) + 0xFC * truth_factor
true_G = G * (1 - truth_factor) + 0x89 * truth_factor
true_B = B * (1 - truth_factor) + 0xAC * truth_factor
```

Primer poziva programa:

```
./optimize-perception propaganda.bmp the-truth.bmp 0.7
```

sa `truth_factor=0.7`, zadatim ulazom u datoteci `propaganda.bmp` i očekivanim izlazom u datoteci `the-truth.bmp`.

Ograničenja zadatka:

- svi nizovi moraju biti **dinamički alocirani**, upotrebom `calloc` funkcije
- sva dinamički zauzeta memorija mora biti oslobođena na odgovarajući način

4.1.1 Primer rešenja

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define HEADER_SIZE 54
#define WIDTH_POS 18
#define HEIGHT_POS 22

typedef struct pixel_st {
    unsigned char B;
    unsigned char G;
    unsigned char R;
} PIXEL;


FILE *safe_fopen(char *name, char *mode);
void *safe_calloc(size_t nmemb, size_t size);
unsigned examine_image_header(FILE *in, unsigned char *header);
void read_image(FILE *in, PIXEL *pixels, unsigned image_size);
void write_image(FILE *out, unsigned char *header, PIXEL *pixels,
unsigned image_size);
void improve_truth(PIXEL *pixels, unsigned image_size, double truth_factor);


int main(int num_args, char **args) {
    if(num_args != 4) {
        printf("USAGE: %s in.bmp out.bmp truth_factor\n", args[0]);
        exit(1);
    }
```

```
    FILE *in = safe_fopen(args[1], "rb");
    FILE *out = safe_fopen(args[2], "wb");
    double truth_factor = atof(args[3]);

    unsigned char *header = safe_calloc(HEADER_SIZE, sizeof(unsigned char));
    unsigned image_size = examine_image_header(in, header);

    PIXEL *pixels = safe_calloc(image_size, sizeof(PIXEL));
    read_image(in, pixels, image_size);

    improve_truth(pixels, image_size, truth_factor);

    write_image(out, header, pixels, image_size);

    fclose(in);
    fclose(out);

    free(header);
    free(pixels);

    return 0;
}

FILE *safe_fopen(char *name, char *mode) {
    FILE *pf = fopen(name, mode);

    if(pf == NULL) {
        printf("File %s could not be opened!\n", name);
        exit(EXIT_FAILURE);
    }

    return pf;
}

void *safe_calloc(size_t nmemb, size_t size) {
    void *new = calloc(nmemb, size);

    if(new == NULL) {
        puts("Not enough RAM!");
        exit(EXIT_FAILURE);
    }

    return new;
}

unsigned examine_image_header(FILE *in, unsigned char *header) {
    fread(header, sizeof(unsigned char), HEADER_SIZE, in);

    unsigned width = *(unsigned *) &header[WIDTH_POS];
    unsigned height = *(unsigned *) &header[HEIGHT_POS];

    return width * height;
}

void read_image(FILE *in, PIXEL *pixels, unsigned image_size) {
    fread(pixels, sizeof(PIXEL), image_size, in);
```

```
}

void write_image(FILE *out, unsigned char *header, PIXEL *pixels,
unsigned image_size) {
    fwrite(header, sizeof(unsigned char), HEADER_SIZE, out);
    fwrite(pixels, sizeof(PIXEL), image_size, out);
}

void improve_truth(PIXEL *pixels, unsigned image_size, double truth_factor) {
    unsigned i;
    for(i=0; i<image_size; i++) {
        pixels[i].R = pixels[i].R * (1 - truth_factor) + 0xFC * truth_factor;
        pixels[i].G = pixels[i].G * (1 - truth_factor) + 0x89 * truth_factor;
        pixels[i].B = pixels[i].B * (1 - truth_factor) + 0xAC * truth_factor;
    }
}
```

4.2 Zadatak „Najpovoljnije meso”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteka učitati podatke u jednostruko spregnutu listu, gde struktura `meso_st` sadrži sledeća polja:

- Skraćena oznaka grada (jedna reč, tačno 2 karaktera)
- Naziv vrste mesa (jedna reč, do 20 karaktera)
- Cena mesa po kilogramu (realan broj)

Naravno, struktura `meso_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Na osnovu zadate vrste mesa `vrsta_mesa` upisati podatke iz formirane liste u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `meso_st`:

- Cena mesa po kilogramu (zaokružena na 2 decimale, koristiti `"%6.2f"` format specifikator)
- Skraćena oznaka grada
- Naziv vrste mesa

i potom u istu izlaznu datoteku upisati informaciju u kom gradu je zadata `vrsta_mesa` najpovoljnija.

Primer poziva:

```
./povoljno_meso junetina cene_mesa.txt analiza.txt
```

sa `vrsta_mesa=junetina`, zadatim ulazom u datoteci `cene_mesa.txt`:

```
NS svinjetina 429.99
NS junetina   649.99
BG junetina   639.99
BG svinjetina 389.99
NI svinjetina 439.99
NI junetina   639.75
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
429.99 NS svinjetina
649.99 NS junetina
639.99 BG junetina
389.99 BG svinjetina
439.99 NI svinjetina
639.75 NI junetina

Najpovoljnija 'junetina' je:
639.75 NI junetina
```

Primer poziva kada tražena vrsta mesa ne postoji:

```
./povoljno_meso teletina stare_cene_mesa.txt analiza.txt
```

sa `vrsta_mesa=teletina`, zadatim ulazom u datoteci `stare_cene_mesa.txt`:

```
NS svinjetina 29.99
BG svinjetina 35.99
NI svinjetina 31.99
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
 29.99 NS svinjetina
 35.99 BG svinjetina
 31.99 NI svinjetina

Nigde nema 'teletina'!
```

4.2.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV_GRAD 2+1
#define MAX_NAZIV_VRSTA_MESA 20+1

typedef struct meso_st {
    char grad[MAX_NAZIV_GRAD];
    char vrsta_mesa[MAX_NAZIV_VRSTA_MESA];
    double cena;
    struct meso_st *next;
} MESO;

void init_list(MESO **head) {
```

```
    *head = NULL;
}

void add_to_list(MESO *new, MESO **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

MESO *create_new_item(char grad[], char vrsta_mesa[], double cena) {
    MESO *new = (MESO *)malloc(sizeof(MESO));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->grad, grad);
    strcpy(new->vrsta_mesa, vrsta_mesa);
    new->cena = cena;

    new->next = NULL;

    return new;
}

void read_list_from(FILE *in, MESO **head) {
    char tmp_grad[MAX_NAZIV_GRAD];
    char tmp_vrsta_mesa[MAX_NAZIV_VRSTA_MESA];
    double tmp_cena;

    while(fscanf(in, "%s %s %lf", tmp_grad, tmp_vrsta_mesa, &tmp_cena) != EOF) {
        MESO *new = create_new_item(tmp_grad, tmp_vrsta_mesa, tmp_cena);
        add_to_list(new, head);
    }
}

void save_item_to(FILE *out, MESO *x) {
    fprintf(
        out, "%6.2lf %s %s\n",
        x->cena, x->grad, x->vrsta_mesa
    );
}

void save_list_to(FILE *out, MESO *head) {
    if(head != NULL) {
        save_item_to(out, head);
        save_list_to(out, head->next);
    }
}

void destroy_list(MESO **head) {
    if(*head != NULL) {
        destroy_list(&((*head)->next));
        free(*head);
```

```
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

MESO *get_najpovoljnije_meso(MESO *head, char vrsta_mesa[]) {
    if (head == NULL) { // list is empty
        return NULL;
    }

    MESO *best = NULL;
    while(head != NULL) {
        if (strcmp(head->vrsta_mesa, vrsta_mesa) == 0) {
            // Gledamo samo meso koje je odgovarajuce vrste
            if (best == NULL) {
                // Pre ovoga sigurno nije bilo mesa koje je odgovarajuce vrste
                best = head;
            } else if (head->cena < best->cena) {
                // Nadjeno povoljnije meso, koje je odgovarajuce vrste
                best = head;
            }
        }

        head = head->next;
    }

    return best;
}

int main(int arg_num, char *args[]) {
    if (arg_num != 4) {
        printf("USAGE: %s VRSTA_MESA IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *vrsta_mesa = args[1];
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    MESO *head;
    init_list(&head);

    read_list_from(in, &head);
    save_list_to(out, head);

    MESO *best = get_najpovoljnije_meso(head, vrsta_mesa);
```

```
    if (best == NULL) {
        fprintf(out, "\nNigde nema '%s'!\n", vrsta_mesa);
    } else {
        fprintf(out, "\nNajpovoljnija '%s' je:\n", vrsta_mesa);
        save_item_to(out, best);
    }

    destroy_list(&head);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.3 Zadatak „Najpovoljnija igra”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteka učitati podatke u jednostruko spregnutu listu, gde struktura `igra_st` sadrži sledeća polja:

- Naziv igre (jedna reč, do 10 karaktera)
- Naziv žanra (jedna reč, do 8 karaktera)
- Naziv platforme (jedna reč, do 8 karaktera)
- Cena igre (pozitivan realan broj)

Naravno, struktura `igra_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Na osnovu zadate platforme `platforma` i žanra `zanr` upisati podatke iz formirane liste u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `igra_st`:

- Naziv igre (koristiti `"%-10s"` format specifikator)
- Naziv žanra (koristiti `"%-8s"` format specifikator)
- Naziv platforme (koristiti `"%-8s"` format specifikator)
- Cena igre (zaokružena na 2 decimale, koristiti `"%5.2f"` format specifikator)

i potom u istu izlaznu datoteku upisati informaciju koja je igra najpovoljnija za zadatu platformu i žanr.

Primer poziva:

```
./povoljne_igre PC Sandbox igre.txt analiza.txt
```

sa `platforma=PC`, `zanr=Sandbox` i zadatim ulazom u datoteci `igre.txt`:

```
Dota2      MOBA     PC        0.00
Fallout3   RPG      PC       19.99
Minecraft  Sandbox  PC       19.95
Minecraft  Sandbox  PS3      18.99
Minecraft  Sandbox  Android   5.49
Factorio   Sandbox  PC       12.50
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
Dota2      MOBA     PC        0.00
Fallout3   RPG      PC       19.99
Minecraft  Sandbox  PC       19.95
Minecraft  Sandbox  PS3      18.99
Minecraft  Sandbox  Android   5.49
Factorio   Sandbox  PC       12.50

Najpovoljnija 'Sandbox' igra za PC platformu je:
Factorio 12.50
```

Primer poziva kada igra sa zadatim kriterijumima ne postoji:

```
./povoljne_igre Atari MOBA stare_igre.txt analiza.txt
```

sa `platforma=Atari`, `zanr=MOBA`, zadatim ulazom u datoteci `stare_igre.txt`:

```
Pong       Arcade   Atari     0.00
Breakout   Arcade   Atari     1.20
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
Pong       Arcade   Atari     0.00
Breakout   Arcade   Atari     1.20

Za Atari platformu ne postoje 'MOBA' igre!
```

4.3.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_IGRA 10+1
#define MAX_ZANR 8+1
#define MAX_PLATFORMA 8+1

typedef struct igra_st {
    char igra[MAX_IGRA];
    char zanr[MAX_ZANR];
    char platforma[MAX_PLATFORMA];
    double cena;
    struct igra_st *next;
} IGRA;

void init_list(IGRA **head) {
```

```
    *head = NULL;
}

void add_to_list(IGRA *new, IGRA **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

IGRA *create_new_item(char igra[], char zanr[], char platforma[], double cena) {
    IGRA *new = (IGRA *)malloc(sizeof(IGRA));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->igra, igra);
    strcpy(new->zanr, zanr);
    strcpy(new->platforma, platforma);
    new->cena = cena;

    new->next = NULL;

    return new;
}

void read_list_from(FILE *in, IGRA **head) {
    char igra[MAX_IGRA];
    char zanr[MAX_ZANR];
    char platforma[MAX_PLATFORMA];
    double cena;

    while(fscanf(in, "%s %s %s %lf", igra, zanr, platforma, &cena) != EOF) {
        IGRA *new = create_new_item(igra, zanr, platforma, cena);
        add_to_list(new, head);
    }
}

void save_item_to(FILE *out, IGRA *x) {
    fprintf(
        out, "%-10s %-8s %-8s %5.2f\n",
        x->igra, x->zanr, x->platforma, x->cena
    );
}

void save_list_to(FILE *out, IGRA *head) {
    if(head != NULL) {
        save_item_to(out, head);
        save_list_to(out, head->next);
    }
}

void destroy_list(IGRA **head) {
    if(*head != NULL) {
```

```
        destroy_list(&((*head)->next));
        free(*head);
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

IGRA *get_najpovoljnija_igra(IGRA *head, char zanr[], char platforma[]) {
    if (head == NULL) { // list is empty
        return NULL;
    }

    IGRA *best = NULL;
    while(head != NULL) {
        if (strcmp(head->zanr, zanr) == 0 &&
            strcmp(head->platforma, platforma) == 0) {
            // Gledamo samo igre koje su OK
            if (best == NULL) {
                // Pre ovoga sigurno nije bilo igre koja je OK
                best = head;
            } else if (head->cena < best->cena) {
                // Nadjena povoljnija igra, koja je OK
                best = head;
            }
        }

        head = head->next;
    }

    return best;
}

int main(int arg_num, char *args[]) {
    if (arg_num != 5) {
        printf("USAGE: %s PLATFORMA ZANR IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *platforma = args[1];
    char *zanr = args[2];
    char *in_filename = args[3];
    char *out_filename = args[4];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    IGRA *head;
    init_list(&head);
```

```
    read_list_from(in, &head);
    save_list_to(out, head);

    IGRA *best = get_najpovoljnija_igra(head, zanr, platforma);
    if (best == NULL) {
        fprintf(
            out, "\nZa %s platformu ne postoje '%s' igre!\n",
            platforma, zanr
        );
    } else {
        fprintf(
            out, "\nNajpovoljnija '%s' igra za %s platformu je:\n%s %.2f\n",
            zanr, platforma, best->igra, best->cena
        );
    }

    destroy_list(&head);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.4 Zadatak „Spektroskopija: teški metali u piću”

Autor zadatka: Petar Marić <*petarmaric@uns.ac.rs*>

Iz zadate ulazne datoteke učitati podatke u jednostruko spregnutu listu, gde struktura `element_st` sadrži sledeća polja:

- hemijski simbol (jedna reč, do 2 karaktera)
- ime (jedna reč, do 20 karaktera)
- atomska težina (prirodan broj)
- vrsta (jedna reč, do 20 karaktera)

Naravno, struktura `element_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Na osnovu zadate vrste hemijskog elementa `vrsta` iz formirane liste upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `element_st`:

- hemijski simbol (koristiti `"%-2s"` format specifikator)
- atomska težina (koristiti `"%3u"` format specifikator)
- vrsta

i potom u istu izlaznu datoteku upisati informaciju o najtežem hemijskom elementu koji pripada zadatoj `vrsti`.

Primer poziva:

```
./najtezi metal uzorak_vino.txt analiza.txt
```

sa `vrsta=metal`, zadatim ulazom u datoteci `uzorak_vino.txt`:

```
C  ugljenik  12 nemetal
H  vodonik    1 nemetal
O  kiseonik  16 nemetal
Ca kalcijum  40 metal
S  sumpor    32 nemetal
Pb olovo    207 metal
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
C   12 nemetal
H    1 nemetal
O   16 nemetal
Ca  40 metal
S   32 nemetal
Pb 207 metal

Najtezi metal je olovo (Pb), atomska tezina 207
```

Primer poziva kada tražena vrsta hemijskog elementa nije pronađena:

```
./najtezi halogen uzorak_voda.txt zagadjenost.txt
```

sa `vrsta=halogen`, zadatim ulazom u datoteci `uzorak_voda.txt`:

```
H vodonik   1 nemetal
O kiseonik 16 nemetal
```

i očekivanim izlazom u datoteci `zagadjenost.txt`:

```
H   1 nemetal
O  16 nemetal

U uzorku nije pronadjen nijedan halogen!
```

4.4.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIMBOL 2+1
#define MAX_IME 20+1
#define MAX_VRSTA 20+1

typedef struct element_st {
    char simbol[MAX_SIMBOL];
    char ime[MAX_IME];
    unsigned tezina;
```

```
    char vrsta[MAX_VRSTA];

    struct element_st *next;
} ELEMENT;

void init_list(ELEMENT **head) {
    *head = NULL;
}

void add_to_list(ELEMENT *new, ELEMENT **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

ELEMENT *create_new_item(char simbol[], char ime[], unsigned tezina,
char vrsta[]) {
    ELEMENT *new = (ELEMENT *)malloc(sizeof(ELEMENT));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->simbol, simbol);
    strcpy(new->ime, ime);
    new->tezina = tezina;
    strcpy(new->vrsta, vrsta);

    new->next = NULL;

    return new;
}

void read_list_from(FILE *in, ELEMENT **head) {
    char simbol[MAX_SIMBOL];
    char ime[MAX_IME];
    unsigned tezina;
    char vrsta[MAX_VRSTA];

    while(fscanf(in, "%s %s %u %s", simbol, ime, &tezina, vrsta) != EOF) {
        ELEMENT *new = create_new_item(simbol, ime, tezina, vrsta);
        add_to_list(new, head);
    }
}

void save_item_to(FILE *out, ELEMENT *x) {
    fprintf(
        out, "%-2s %3u %s\n",
        x->simbol, x->tezina, x->vrsta
    );
}

void save_list_to(FILE *out, ELEMENT *head) {
    if(head != NULL) {
```

```
        save_item_to(out, head);
        save_list_to(out, head->next);
    }
}

void destroy_list(ELEMENT **head) {
    if(*head != NULL) {
        destroy_list(&((*head)->next));
        free(*head);
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

ELEMENT *get_najtezi_element(ELEMENT *head, char vrsta[]) {
    if (head == NULL) { // list is empty
        return NULL;
    }

    ELEMENT *best = NULL;
    while(head != NULL) {
        if (strcmp(head->vrsta, vrsta) == 0) {
            // Gledamo samo hem. element koji je odgovarajuce vrste
            if (best == NULL) {
                // Pre ovoga nije bilo hem. elemenata koji su odgovarajuce vrste
                best = head;
            } else if (head->tezina > best->tezina) {
                // Nadjen tezi hem. element, koji je odgovarajuce vrste
                best = head;
            }
        }

        head = head->next;
    }

    return best;
}

int main(int arg_num, char *args[]) {
    if (arg_num != 4) {
        printf("USAGE: %s VRSTA IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *vrsta = args[1];
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
```

```
    FILE *out = safe_fopen(out_filename, "w", 2);

    ELEMENT *head;
    init_list(&head);

    read_list_from(in, &head);
    save_list_to(out, head);

    ELEMENT *best = get_najtezi_element(head, vrsta);
    if (best == NULL) {
        fprintf(out, "\nU uzorku nije pronadjen nijedan %s!\n", vrsta);
    } else {
        fprintf(
            out, "\nNajtezi %s je %s (%s), atomska tezina %u\n",
            best->vrsta, best->ime, best->simbol, best->tezina
        );
    }

    destroy_list(&head);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.5 Zadatak „Prosečna cena goriva”

Autor zadatka: Petar Marić <*petarmaric@uns.ac.rs*>

Iz zadate ulazne datoteke učitati podatke u jednostruko spregnutu listu, gde struktura `gorivo_st` sadrži sledeća polja:

- skraćena oznaka grada (jedna reč, tačno 2 karaktera)
- tip goriva (jedna reč, do 10 karaktera)
- cena goriva (pozitivan realan broj)

Naravno, struktura `gorivo_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Na osnovu zadatog tipa goriva `tip_goriva` (`gas`, `benzin` ili `dizel`) iz formirane liste upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `gorivo_st`:

- cena (zaokružena na 2 decimale, koristiti `"%6.2f"` format specifikator)
- skraćena oznaka grada
- tip goriva

i potom u istu izlaznu datoteku upisati informaciju koja je prosečna cena zadatog tipa goriva (zaokružena na 2 decimale, koristiti `"%6.2f"` format specifikator).

Primer poziva:

```
./gorivo benzin cene.txt izvestaj.txt
```

sa `tip_goriva=benzin` i zadatim ulazom u datoteci `cene.txt`:

```
NS gas     71.90
NS benzin 155.20
NS dizel  165.15
NI gas     69.30
NI benzin 155.30
NI dizel  165.20
BG gas     70.20
BG benzin 155.50
BG dizel  165.40
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
 71.90 NS gas
155.20 NS benzin
165.15 NS dizel
 69.30 NI gas
155.30 NI benzin
165.20 NI dizel
 70.20 BG gas
155.50 BG benzin
165.40 BG dizel

AVG = 155.33
```

Primer poziva kada gorivo sa zadatim kriterijumima ne postoji:

```
./gorivo gas mala_pumpa.txt izvestaj.txt
```

sa `tip_goriva=gas` i zadatim ulazom u datoteci `mala_pumpa.txt`:

```
SA benzin 196.90
PE dizel  191.60
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
196.90 SA benzin
191.60 PE dizel

NOT FOUND!
```

4.5.1 Primer rešenja

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_GRAD 2+1
#define MAX_TIP_GORIVA 10+1
```

```c
#define ERROR_MAGIC_NUM -666.999

typedef struct gorivo_st {
    char grad[MAX_GRAD];
    char tip_goriva[MAX_TIP_GORIVA];
    double cena;

    struct gorivo_st *next;
} GORIVO;

void init_list(GORIVO **head) {
    *head = NULL;
}

void add_to_list(GORIVO *new, GORIVO **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

GORIVO *create_new_item(char grad[], char tip_goriva[], double cena) {
    GORIVO *new = (GORIVO *)malloc(sizeof(GORIVO));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->grad, grad);
    strcpy(new->tip_goriva, tip_goriva);
    new->cena = cena;

    new->next = NULL;

    return new;
}

void read_list_from(FILE *in, GORIVO **head) {
    char grad[MAX_GRAD];
    char tip_goriva[MAX_TIP_GORIVA];
    double cena;

    while(fscanf(in, "%s %s %lf", grad, tip_goriva, &cena) != EOF) {
        GORIVO *new = create_new_item(grad, tip_goriva, cena);
        add_to_list(new, head);
    }
}

void save_item_to(FILE *out, GORIVO *x) {
    fprintf(
        out, "%6.2f %s %s\n",
        x->cena, x->grad, x->tip_goriva
    );
}
```

```c
void save_list_to(FILE *out, GORIVO *head) {
    if(head != NULL) {
        save_item_to(out, head);
        save_list_to(out, head->next);
    }
}

void destroy_list(GORIVO **head) {
    if(*head != NULL) {
        destroy_list(&((*head)->next));
        free(*head);
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

int count_gorivo(GORIVO *head, char tip_goriva[]) {
    int count = 0;

    while(head != NULL) {
        if (strcmp(head->tip_goriva, tip_goriva) == 0) {
            count++;
        }

        head = head->next;
    }

    return count;
}

double sum_gorivo(GORIVO *head, char tip_goriva[]) {
    double sum = 0;

    while(head != NULL) {
        if (strcmp(head->tip_goriva, tip_goriva) == 0) {
            sum += head->cena;
        }

        head = head->next;
    }

    return sum;
}

double avg_gorivo(GORIVO *head, char tip_goriva[]) {
    int count = count_gorivo(head, tip_goriva);
    if (count == 0) {
        return ERROR_MAGIC_NUM;
```

```
    } else {
        return sum_gorivo(head, tip_goriva) / count;
    }
}

int main(int arg_num, char *args[]) {
    if (arg_num != 4) {
        printf("USAGE: %s TIP_GORIVA IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *tip_goriva = args[1];
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    GORIVO *head;
    init_list(&head);

    read_list_from(in, &head);
    save_list_to(out, head);

    double avg = avg_gorivo(head, tip_goriva);
    if (avg == ERROR_MAGIC_NUM) {
        fprintf(out, "\nNOT FOUND!\n");
    } else {
        fprintf(out, "\nAVG = %6.2lf\n", avg);
    }

    destroy_list(&head);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.6 Zadatak „prinesi.com”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati podatke u jednostruko spregnutu listu, gde struktura `restoran_st` sadrži sledeća polja:

- naziv restorana (jedna reč, do 10 karaktera)
- vrsta kuhinje (jedna reč, do 20 karaktera)
- prosečna ocena korisnika (pozitivan realan broj)

Naravno, struktura `restoran_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Na osnovu zadate vrste kuhinje `vrsta` iz formirane liste upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `restoran_st`:

- prosečna ocena korisnika (zaokružena na 1 decimalu, koristiti `"%3.1f"` format specifikator)
- naziv restorana (koristiti `"%-10s"` format specifikator)
- vrsta kuhinje

i potom u istu izlaznu datoteku upisati informaciju o najbolje ocenjenom restoranu koji služi hranu zadate `vrste` kuhinje.

Primer poziva:

```
./restorani italijanski novi-sad.txt izvestaj.txt
```

sa `vrsta=italijanski` i zadatim ulazom u datoteci `novi-sad.txt`:

```
Cremansa   italijanski  4.3
Sekunda    italijanski  3.9
Inimigos   americki     4.5
LaCattura  italijanski  4.7
FutureFood americki     4.4
Eva        srpski       4.8
Kokoda     srpski       4.3
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
4.3 Cremansa   italijanski
3.9 Sekunda    italijanski
4.5 Inimigos   americki
4.7 LaCattura  italijanski
4.4 FutureFood americki
4.8 Eva        srpski
4.3 Kokoda     srpski

Najbolje ocenjen italijanski restoran u gradu je:
4.7 LaCattura  italijanski
```

Primer poziva kada restoran sa zadatim kriterijumima ne postoji:

```
./restorani kineski kraljevo.txt izvestaj.txt
```

sa `vrsta=kineski` i zadatim ulazom u datoteci `kraljevo.txt`:

```
PizzaSlow  italijanski  4.5
SaleDjodjo italijanski  4.4
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
4.5 PizzaSlow  italijanski
4.4 SaleDjodjo italijanski

U gradu ne postoji kineski restoran!
```

4.6.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV 10+1
#define MAX_VRSTA 20+1

typedef struct restoran_st {
    char naziv[MAX_NAZIV];
    char vrsta[MAX_VRSTA];
    double ocena;

    struct restoran_st *next;
} RESTORAN;

void init_list(RESTORAN **head) {
    *head = NULL;
}

void add_to_list(RESTORAN *new, RESTORAN **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

RESTORAN *create_new_item(char naziv[], char vrsta[], double ocena) {
    RESTORAN *new = (RESTORAN *)malloc(sizeof(RESTORAN));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->naziv, naziv);
    strcpy(new->vrsta, vrsta);
    new->ocena = ocena;

    new->next = NULL;

    return new;
}

void read_list_from(FILE *in, RESTORAN **head) {
    char naziv[MAX_NAZIV];
    char vrsta[MAX_VRSTA];
    double ocena;

    while(fscanf(in, "%s %s %lf", naziv, vrsta, &ocena) != EOF) {
        RESTORAN *new = create_new_item(naziv, vrsta, ocena);
        add_to_list(new, head);
    }
}

void save_item_to(FILE *out, RESTORAN *x) {
```

```
    fprintf(
        out, "%3.1f %-10s %s\n",
        x->ocena, x->naziv, x->vrsta
    );
}

void save_list_to(FILE *out, RESTORAN *head) {
    if(head != NULL) {
        save_item_to(out, head);
        save_list_to(out, head->next);
    }
}

void destroy_list(RESTORAN **head) {
    if(*head != NULL) {
        destroy_list(&((*head)->next));
        free(*head);
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

RESTORAN *get_najbolji_restoran(RESTORAN *head, char vrsta[]) {
    if (head == NULL) { // list is empty
        return NULL;
    }

    RESTORAN *best = NULL;
    while(head != NULL) {
        if (strcmp(head->vrsta, vrsta) == 0) {
            // Gledamo samo restoran koji je odgovarajuce vrste
            if (best == NULL) {
                // Pre ovoga nije bilo restorana koji su odgovarajuce vrste
                best = head;
            } else if (head->ocena > best->ocena) {
                // Nadjen bolji restoran, koji je odgovarajuce vrste
                best = head;
            }
        }

        head = head->next;
    }

    return best;
}

int main(int arg_num, char *args[]) {
    if (arg_num != 4) {
        printf("USAGE: %s VRSTA IN_FILENAME OUT_FILENAME\n", args[0]);
```

```
        exit(11);
    }

    char *vrsta = args[1];
    char *in_filename = args[2];
    char *out_filename = args[3];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    RESTORAN *head;
    init_list(&head);

    read_list_from(in, &head);
    save_list_to(out, head);

    RESTORAN *best = get_najbolji_restoran(head, vrsta);
    if (best == NULL) {
        fprintf(out, "\nU gradu ne postoji %s restoran!\n", vrsta);
    } else {
        fprintf(
            out, "\nNajbolje ocenjen %s restoran u gradu je:\n",
            best->vrsta
        );
        save_item_to(out, best);
    }

    destroy_list(&head);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.7 Zadatak „Cena pizze u svetu”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati podatke u jednostruko spregnutu listu, gde struktura `pizza_st` sadrži sledeća polja:

- Cena (prirodan broj)
- Skraćena oznaka države (jedna reč, tačno 3 karaktera)
- Skraćena oznaka grada (jedna reč, tačno 2 karaktera)
- Naziv pizze (jedna reč, do 20 karaktera)

Naravno, struktura `pizza_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Na osnovu zadate oznake države `drzava` i naziva pizze `naziv_pizze` iz formirane liste upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja

strukture pizza_st:

- Cena (koristiti "%4d" format specifikator)
- Skraćena oznaka države
- Skraćena oznaka grada
- Naziv pizze

i potom u istu izlaznu datoteku upisati informaciju koja je prosečna cena pizze (zaokružena na 2 decimale) za zadatu državu i vrstu pizze.

Primer poziva:

```
./zad SRB Capricciosa pizze.txt analiza.txt
```

sa `drzava=SRB`, `naziv_pizze=Capricciosa` i zadatim ulazom u datoteci `pizze.txt`:

```
 503 SRB NS Capricciosa
 530 SRB NS Madjarica
1270 SRB NS Prosciutto
 990 ITA NA Margharita
1020 ITA NA Capricciosa
1120 ITA NA Prosciutto
 540 SRB BG Capricciosa
 620 SRB BG Madjarica
 320 USA NY YankeeSlice
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
 503 SRB NS Capricciosa
 530 SRB NS Madjarica
1270 SRB NS Prosciutto
 990 ITA NA Margharita
1020 ITA NA Capricciosa
1120 ITA NA Prosciutto
 540 SRB BG Capricciosa
 620 SRB BG Madjarica
 320 USA NY YankeeSlice

avg = 521.50
```

Primer poziva kada pizza sa zadatim kriterijumima ne postoji:

```
./zad SRB Margharita ns_pizze.txt analiza.txt
```

sa `drzava=SRB`, `vrsta_pizze=Margharita` i zadatim ulazom u datoteci `ns_pizze.txt`:

```
 530 SRB NS Madjarica
1270 SRB NS Prosciutto
 510 SRB NS Capricciosa
```

i očekivanim izlazom u datoteci analiza.txt:

```
 530 SRB NS Madjarica
1270 SRB NS Prosciutto
 510 SRB NS Capricciosa

NOT FOUND!
```

4.7.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_DRZAVA 3+1
#define MAX_GRAD 2+1
#define MAX_NAZIV 20+1

#define ERROR_MAGIC_NUM -666.999

typedef struct pizza_st {
    int cena;
    char drzava[MAX_DRZAVA];
    char grad[MAX_GRAD];
    char naziv[MAX_NAZIV];
    struct pizza_st *next;
} PIZZA;

void init_list(PIZZA **head) {
    *head = NULL;
}

void add_to_list(PIZZA *new, PIZZA **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

PIZZA *create_new_item(int cena, char drzava[], char grad[], char naziv[]) {
    PIZZA *new = (PIZZA *)malloc(sizeof(PIZZA));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    new->cena = cena;
    strcpy(new->drzava, drzava);
    strcpy(new->grad, grad);
    strcpy(new->naziv, naziv);

    new->next = NULL;

    return new;
}
```

```c
void read_list_from(FILE *in, PIZZA **head) {
    int cena;
    char drzava[MAX_DRZAVA];
    char grad[MAX_GRAD];
    char naziv[MAX_NAZIV];

    while(fscanf(in, "%d %s %s %s", &cena, drzava, grad, naziv) != EOF) {
        PIZZA *new = create_new_item(cena, drzava, grad, naziv);
        add_to_list(new, head);
    }
}

void save_item_to(FILE *out, PIZZA *x) {
    fprintf(
        out, "%4d %s %s %s\n",
        x->cena, x->drzava, x->grad, x->naziv
    );
}

void save_list_to(FILE *out, PIZZA *head) {
    if(head != NULL) {
        save_item_to(out, head);
        save_list_to(out, head->next);
    }
}

void destroy_list(PIZZA **head) {
    if(*head != NULL) {
        destroy_list(&((*head)->next));
        free(*head);
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

int count_pizza(PIZZA *head, char drzava[], char naziv[]) {
    int count = 0;

    while(head != NULL) {
        if (strcmp(head->drzava, drzava) == 0 &&
            strcmp(head->naziv, naziv) == 0) {
            count++;
        }

        head = head->next;
    }

    return count;
}
```

```
int sum_pizza(PIZZA *head, char drzava[], char naziv[]) {
    int sum = 0;

    while(head != NULL) {
        if (strcmp(head->drzava, drzava) == 0 &&
            strcmp(head->naziv, naziv) == 0) {
            sum += head->cena;
        }

        head = head->next;
    }

    return sum;
}

double avg_pizza(PIZZA *head, char drzava[], char naziv[]) {
    double count = count_pizza(head, drzava, naziv);
    if (count == 0) {
        return ERROR_MAGIC_NUM;
    } else {
        return sum_pizza(head, drzava, naziv) / count;
    }
}

int main(int arg_num, char *args[]) {
    if (arg_num != 5) {
        printf("USAGE: %s DRZAVA NAZIV IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *drzava = args[1];
    char *naziv = args[2];
    char *in_filename = args[3];
    char *out_filename = args[4];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    PIZZA *head;
    init_list(&head);

    read_list_from(in, &head);
    save_list_to(out, head);

    double avg = avg_pizza(head, drzava, naziv);
    if (avg == ERROR_MAGIC_NUM) {
        fprintf(out, "\nNOT FOUND!\n");
    } else {
        fprintf(out, "\navg = %.2lf\n", avg);
    }

    destroy_list(&head);

    fclose(in);
    fclose(out);
```

```
    return 0;
}
```

4.8 Zadatak „Pizza Pizza Pizza”

Autor zadatka: Petar Marić <*petarmaric@uns.ac.rs*>

Iz zadate ulazne datoteke učitati podatke u jednostruko spregnutu listu, gde struktura `pizza_st` sadrži sledeća polja:

- Cena (prirodan broj)
- Skraćena oznaka države (jedna reč, tačno 3 karaktera)
- Skraćena oznaka grada (jedna reč, tačno 2 karaktera)
- Naziv pizze (jedna reč, do 20 karaktera)

Naravno, struktura `pizza_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Na osnovu zadate oznake države `drzava` i naziva pizze `naziv_pizze` iz formirane liste upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture pizza_st:

- Cena (koristiti `"%4d"` format specifikator)
- Skraćena oznaka države
- Skraćena oznaka grada
- Naziv pizze

i potom u istu izlaznu datoteku upisati informaciju u koliko gradova se može naći pizza pod zadatim kriterijumima.

Primer poziva:

```
./zad SRB Capricciosa pizze.txt analiza.txt
```

sa `drzava=SRB`, `naziv_pizze=Capricciosa` i zadatim ulazom u datoteci `pizze.txt`:

```
 510 SRB NS Capricciosa
 530 SRB NS Madjarica
1270 SRB NS Prosciutto
 990 ITA NA Margharita
1020 ITA NA Capricciosa
1120 ITA NA Prosciutto
 540 SRB BG Capricciosa
 620 SRB BG Madjarica
 320 USA NY YankeeSlice
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
 510 SRB NS Capricciosa
 530 SRB NS Madjarica
1270 SRB NS Prosciutto
 990 ITA NA Margharita
1020 ITA NA Capricciosa
1120 ITA NA Prosciutto
 540 SRB BG Capricciosa
 620 SRB BG Madjarica
 320 USA NY YankeeSlice

count = 2
```

4.8.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_DRZAVA 3+1
#define MAX_GRAD 2+1
#define MAX_NAZIV 20+1

typedef struct pizza_st {
    int cena;
    char drzava[MAX_DRZAVA];
    char grad[MAX_GRAD];
    char naziv[MAX_NAZIV];
    struct pizza_st *next;
} PIZZA;

void init_list(PIZZA **head) {
    *head = NULL;
}

void add_to_list(PIZZA *new, PIZZA **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

PIZZA *create_new_item(int cena, char drzava[], char grad[], char naziv[]) {
    PIZZA *new = (PIZZA *)malloc(sizeof(PIZZA));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    new->cena = cena;
    strcpy(new->drzava, drzava);
    strcpy(new->grad, grad);
    strcpy(new->naziv, naziv);

    new->next = NULL;
```

```
    return new;
}

void read_list_from(FILE *in, PIZZA **head) {
    int cena;
    char drzava[MAX_DRZAVA];
    char grad[MAX_GRAD];
    char naziv[MAX_NAZIV];

    while(fscanf(in, "%d %s %s %s", &cena, drzava, grad, naziv) != EOF) {
        PIZZA *new = create_new_item(cena, drzava, grad, naziv);
        add_to_list(new, head);
    }
}

void save_item_to(FILE *out, PIZZA *x) {
    fprintf(
        out, "%4d %s %s %s\n",
        x->cena, x->drzava, x->grad, x->naziv
    );
}

void save_list_to(FILE *out, PIZZA *head) {
    if(head != NULL) {
        save_item_to(out, head);
        save_list_to(out, head->next);
    }
}

void destroy_list(PIZZA **head) {
    if(*head != NULL) {
        destroy_list(&((*head)->next));
        free(*head);
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

int count_pizza(PIZZA *head, char drzava[], char naziv[]) {
    int count = 0;

    while(head != NULL) {
        if (strcmp(head->drzava, drzava) == 0 &&
            strcmp(head->naziv, naziv) == 0) {
            count++;
        }

        head = head->next;
```

```
    }

    return count;
}

int main(int arg_num, char *args[]) {
    if (arg_num != 5) {
        printf("USAGE: %s DRZAVA NAZIV IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *drzava = args[1];
    char *naziv = args[2];
    char *in_filename = args[3];
    char *out_filename = args[4];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    PIZZA *head;
    init_list(&head);

    read_list_from(in, &head);
    save_list_to(out, head);

    fprintf(out, "\ncount = %d\n", count_pizza(head, drzava, naziv));

    destroy_list(&head);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.9 Zadatak „Prepoznavanje oblika”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Transformati svaki pixel date slike na sledeći način:

crveni kanal: primeniti `OR` bitwise operator nad ovim i zelenim kanalom

zeleni kanal: pomeriti udesno bite koji pripadaju ovom kanalu za ukupno `offset` podeoka

`offset` je ceo broj koji se unosi kao argument komandne linije

plavi kanal: obrnuti bite koji pripadaju ovom kanalu

Primer poziva programa:

```
./feature-detection original.bmp processed-low.bmp 1
```

sa `offset=1`, zadatim ulazom u datoteci `original.bmp`:

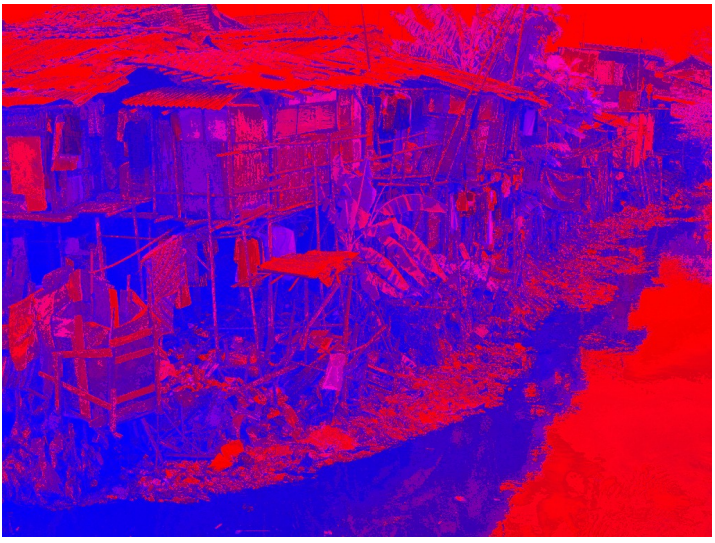
i očekivanim izlazom u datoteci `processed-low.bmp`:

Primer poziva programa:

```
./feature-detection original.bmp processed-high.bmp 7
```

sa `offset=7`, zadatim ulazom u datoteci `original.bmp`:

i očekivanim izlazom u datoteci `processed-high.bmp`:



Izvor slike: https://en.wikipedia.org/wiki/File:Jakarta_slumhome_2.jpg (CC BY 2.0)

Ograničenja zadatka:

- zadatak rešiti upotrebom **jednostruko spregnute liste**
- sva dinamički zauzeta memorija mora biti oslobođena na odgovarajući način

4.9.1 Primer rešenja

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define HEADER_SIZE 54
#define WIDTH_POS 18
#define HEIGHT_POS 22

typedef struct pixel_st {
    unsigned char B;
    unsigned char G;
    unsigned char R;
} PIXEL;

typedef struct node_st {
    PIXEL data;
    struct node_st *next;
} NODE;


FILE *safe_fopen(char *name, char *mode);
void init_list(NODE **head, NODE **tail);
NODE *create_node(PIXEL p);
void add_on_end(NODE **head, NODE **tail, PIXEL p);
void delete_list(NODE **head, NODE **tail);
unsigned examine_image_header(FILE *in, unsigned char *header);
void read_image(FILE *in, NODE **head, NODE **tail);
void write_image(FILE *out, unsigned char *header, NODE *head);
void feature_detection(NODE *head, int offset);


int main(int num_args, char **args) {
    if(num_args != 4) {
        printf("USAGE: %s in.bmp out.bmp offset\n", args[0]);
        exit(1);
    }

    FILE *in = safe_fopen(args[1], "rb");
    FILE *out = safe_fopen(args[2], "wb");
    int offset = atoi(args[3]);

    unsigned char header[HEADER_SIZE];
    unsigned image_size = examine_image_header(in, header);

    NODE *head, *tail;
    init_list(&head, &tail);
    read_image(in, &head, &tail);

    feature_detection(head, offset);

    write_image(out, header, head);

    fclose(in);
    fclose(out);

    delete_list(&head, &tail);
```

```
    return 0;
}

FILE *safe_fopen(char *name, char *mode) {
    FILE *pf = fopen(name, mode);

    if(pf == NULL) {
        printf("File %s could not be opened!\n", name);
        exit(EXIT_FAILURE);
    }

    return pf;
}

void init_list(NODE **head, NODE **tail) {
    *head = NULL;
    *tail = NULL;
}

NODE *create_node(PIXEL p) {
    NODE *new = malloc(sizeof(NODE));

    if(new == NULL) {
        puts("Not enough RAM!");
        exit(EXIT_FAILURE);
    }

    new->data = p;
    new->next = NULL;

    return new;
}

void add_on_end(NODE **head, NODE **tail, PIXEL p) {
    NODE *new = create_node(p);

    if (*head != NULL) {
        (*tail)->next = new;
    } else {
        *head = new;
    }

    *tail = new;
}

void delete_list(NODE **head, NODE **tail) {
    NODE *temp = *head;

    while(*head != NULL) {
        temp = *head;
        *head = (*head)->next;

        temp->next = NULL;
        free(temp);
    }
```

```
    *tail = NULL;
}

unsigned examine_image_header(FILE *in, unsigned char *header) {
    fread(header, sizeof(unsigned char), HEADER_SIZE, in);

    unsigned width = *(unsigned *) &header[WIDTH_POS];
    unsigned height = *(unsigned *) &header[HEIGHT_POS];

    return width * height;
}

void read_image(FILE *in, NODE **head, NODE **tail) {
    PIXEL p;

    while (fread(&p, sizeof(PIXEL), 1, in) != 0) {
        add_on_end(head, tail, p);
    }
}

void write_image(FILE *out, unsigned char *header, NODE *head) {
    fwrite(header, sizeof(unsigned char), HEADER_SIZE, out);

    while(head != NULL) {
        fwrite(&(head->data), sizeof(PIXEL), 1, out);

        head = head->next;
    }
}

void feature_detection(NODE *head, int offset) {
    while(head != NULL) {
        head->data.R |= head->data.G;
        head->data.G >>= offset;
        head->data.B = ~head->data.B;

        head = head->next;
    }
}
```

4.10 Zadatak „odnesi.com”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati podatke u binarno stablo, gde struktura `restoran_st` sadrži sledeća polja:

- naziv restorana (jedna reč, do 10 karaktera)
- vrsta kuhinje (jedna reč, do 20 karaktera)
- prosečna ocena korisnika (pozitivan realan broj)

Naravno, struktura `restoran_st` sadrži i polja potrebna za pravilno formiranje binarnog stabla.

Binarno stablo urediti po ključu „prosečna ocena korisnika”, u opadajućem redosledu. Potom iz formiranog binarnog stabla upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `restoran_st`:

- prosečna ocena korisnika (zaokružena na 1 decimalu, koristiti `"%3.1f"` format specifikator)
- naziv restorana (koristiti `"%-10s"` format specifikator)
- vrsta kuhinje

i potom u istu izlaznu datoteku upisati informaciju o najgore ocenjenom restoranu.

Primer poziva programa:

```
./restorani novi-sad.txt izvestaj.txt
```

sa zadatim ulazom u datoteci `novi-sad.txt`:

```
Cremansa   italijanski  4.3
Sekunda    italijanski  3.9
Inimigos   americki     4.5
LaCattura  italijanski  4.7
FutureFood americki     4.4
Eva        srpski       4.8
Kokoda     srpski       4.2
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
4.8 Eva        srpski
4.7 LaCattura  italijanski
4.5 Inimigos   americki
4.4 FutureFood americki
4.3 Cremansa   italijanski
4.2 Kokoda     srpski
3.9 Sekunda    italijanski

Najgore ocenjen restoran u gradu je:
3.9 Sekunda    italijanski
```

Primer poziva programa:

```
./restorani kineski kraljevo.txt izvestaj.txt
```

sa zadatim ulazom u datoteci `kraljevo.txt`:

```
SaleDjodjo italijanski  4.4
PizzaSlow  italijanski  4.5
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
4.5 PizzaSlow  italijanski
4.4 SaleDjodjo italijanski
```

```
Najgore ocenjen restoran u gradu je:
4.4 SaleDjodjo italijanski
```

4.10.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV 10+1
#define MAX_VRSTA 20+1

typedef struct restoran_st {
    char naziv[MAX_NAZIV];
    char vrsta[MAX_VRSTA];
    double ocena;

    struct restoran_st *left;
    struct restoran_st *right;
} RESTORAN;

void init_tree(RESTORAN **root) {
    *root = NULL;
}

void add_to_tree(RESTORAN *new, RESTORAN **root) {
    if(*root == NULL) { // tree is empty
        *root = new;
    } else if (new->ocena >= (*root)->ocena) {
        add_to_tree(new, &((*root)->left));
    } else {
        add_to_tree(new, &((*root)->right));
    }
}

RESTORAN *create_new_item(char naziv[], char vrsta[], double ocena) {
    RESTORAN *new = (RESTORAN *)malloc(sizeof(RESTORAN));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->naziv, naziv);
    strcpy(new->vrsta, vrsta);
    new->ocena = ocena;

    new->left = NULL;
    new->right = NULL;

    return new;
}

void read_tree_from(FILE *in, RESTORAN **root) {
    char naziv[MAX_NAZIV];
    char vrsta[MAX_VRSTA];
    double ocena;
```

```
    while(fscanf(in, "%s %s %lf", naziv, vrsta, &ocena) != EOF) {
        RESTORAN *new = create_new_item(naziv, vrsta, ocena);
        add_to_tree(new, root);
    }
}

void save_item_to(FILE *out, RESTORAN *x) {
    fprintf(
        out, "%3.1f %-10s %s\n",
        x->ocena, x->naziv, x->vrsta
    );
}

void save_tree_to(FILE *out, RESTORAN *root) {
    if(root != NULL) {
        save_tree_to(out, root->left);
        save_item_to(out, root);
        save_tree_to(out, root->right);
    }
}

void destroy_tree(RESTORAN **root) {
    if(*root != NULL) {
        destroy_tree(&((*root)->left));
        destroy_tree(&((*root)->right));
        free(*root);
        *root = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

RESTORAN *get_najgori_restoran(RESTORAN *root) {
    if (root == NULL) { // tree is empty
        return NULL;
    }

    // Stablo je uredjeno po kljucu "ocena", u opadajucem redosledu.
    // Stoga se najgore ocenjen restoran sigurno nalazi u krajnje desnom listu.

    if (root->right == NULL) {
        // ovaj "root" je krajnje desni list, posto nema desno dete
        return root;
    }

    // Trazi dalje, *sigurno* ima jos gorih restorana
    return get_najgori_restoran(root->right);
}
```

```
int main(int arg_num, char *args[]) {
    if (arg_num != 3) {
        printf("USAGE: %s IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *in_filename = args[1];
    char *out_filename = args[2];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    RESTORAN *root;
    init_tree(&root);

    read_tree_from(in, &root);
    save_tree_to(out, root);

    RESTORAN *worst = get_najgori_restoran(root);
    if (worst != NULL) {
        fprintf(out, "\nNajgore ocenjen restoran u gradu je:\n");
        save_item_to(out, worst);
    }

    destroy_tree(&root);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.11 Zadatak „Namirnica sa najviše vitamina C”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati podatke u binarno stablo, gde struktura `namirnica_st` sadrži sledeća polja:

- naziv namirnice (jedna reč, do 13 karaktera)
- količina vitamina C u namirnici (mg/100g)
- vrsta namirnice (jedna reč, do 10 karaktera)

Naravno, struktura `namirnica_st` sadrži i polja potrebna za pravilno formiranje binarnog stabla.

Binarno stablo urediti po ključu „količina vitamina C”, u opadajućem redosledu. Potom iz formiranog binarnog stabla upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `namirnica_st`:

- količina vitamina (koristiti `"%3u"` format specifikator)
- naziv namirnice (koristiti `"%-13s"` format specifikator)

- vrsta namirnice

i potom u istu izlaznu datoteku upisati informaciju o namirnici sa najviše vitamina C.

Primer poziva programa:

```
./analiza namirnice.txt izvestaj.txt
```

sa zadatim ulazom u datoteci `namirnice.txt`:

```
Ananas          9 voce
Bakalar        26 meso
BeliLuk        31 povrce
Brokoli        90 povrce
CrvenaPaprika 190 povrce
Grejpfrut      29 voce
Jagoda         60 voce
Kivi           89 voce
Krompir        20 povrce
Limun          40 voce
Paradajz       10 povrce
PilecaJetra    13 meso
Pomorandza     50 voce
Ribizla       200 voce
Sipurak       426 voce
Spanac         30 povrce
TelecaJetra    36 meso
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
426 Sipurak       voce
200 Ribizla       voce
190 CrvenaPaprika povrce
 90 Brokoli       povrce
 89 Kivi          voce
 60 Jagoda        voce
 50 Pomorandza    voce
 40 Limun         voce
 36 TelecaJetra   meso
 31 BeliLuk       povrce
 30 Spanac        povrce
 29 Grejpfrut     voce
 26 Bakalar       meso
 20 Krompir       povrce
 13 PilecaJetra   meso
 10 Paradajz      povrce
  9 Ananas        voce

Namirnica sa najvise vitamina C je:
426 Sipurak       voce
```

Primer poziva programa kada su ulazni podaci već sortirani u rastućem redosledu:

```
./analiza sorted-asc.txt izvestaj-asc.txt
```

sa zadatim ulazom u datoteci `sorted-asc.txt`:

```
Limun          40 voce
Pomorandza     50 voce
Jagoda         60 voce
Kivi           89 voce
Brokoli        90 povrce
```

i očekivanim izlazom u datoteci `izvestaj-asc.txt`:

```
 90 Brokoli       povrce
 89 Kivi          voce
 60 Jagoda        voce
 50 Pomorandza    voce
 40 Limun         voce

Namirnica sa najvise vitamina C je:
 90 Brokoli       povrce
```

Primer poziva programa kada su ulazni podaci već sortirani u opadajućem redosledu:

```
./analiza sorted-desc.txt izvestaj-desc.txt
```

sa zadatim ulazom u datoteci `sorted-desc.txt`:

```
Bakalar        26 meso
Krompir        20 povrce
PilecaJetra    13 meso
Paradajz       10 povrce
Ananas          9 voce
```

i očekivanim izlazom u datoteci `izvestaj-desc.txt`:

```
 26 Bakalar       meso
 20 Krompir       povrce
 13 PilecaJetra   meso
 10 Paradajz      povrce
  9 Ananas        voce

Namirnica sa najvise vitamina C je:
 26 Bakalar       meso
```

4.11.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV 13+1
#define MAX_VRSTA 10+1

typedef struct namirnica_st {
    char naziv[MAX_NAZIV];
```

```
    unsigned kolicina;
    char vrsta[MAX_VRSTA];

    struct namirnica_st *left;
    struct namirnica_st *right;
} NAMIRNICA;

void init_tree(NAMIRNICA **root) {
    *root = NULL;
}

void add_to_tree(NAMIRNICA *new, NAMIRNICA **root) {
    if(*root == NULL) { // tree is empty
        *root = new;
    } else if (new->kolicina >= (*root)->kolicina) {
        add_to_tree(new, &((*root)->left));
    } else {
        add_to_tree(new, &((*root)->right));
    }
}

NAMIRNICA *create_new_item(char naziv[], unsigned kolicina, char vrsta[]) {
    NAMIRNICA *new = (NAMIRNICA *)malloc(sizeof(NAMIRNICA));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->naziv, naziv);
    new->kolicina = kolicina;
    strcpy(new->vrsta, vrsta);

    new->left = NULL;
    new->right = NULL;

    return new;
}

void read_tree_from(FILE *in, NAMIRNICA **root) {
    char naziv[MAX_NAZIV];
    unsigned kolicina;
    char vrsta[MAX_VRSTA];

    while(fscanf(in, "%s %u %s", naziv, &kolicina, vrsta) != EOF) {
        NAMIRNICA *new = create_new_item(naziv, kolicina, vrsta);
        add_to_tree(new, root);
    }
}

void save_item_to(FILE *out, NAMIRNICA *x) {
    fprintf(
        out, "%3u %-13s %s\n",
        x->kolicina, x->naziv, x->vrsta
    );
}

void save_tree_to(FILE *out, NAMIRNICA *root) {
```

```
    if(root != NULL) {
        save_tree_to(out, root->left);
        save_item_to(out, root);
        save_tree_to(out, root->right);
    }
}

void destroy_tree(NAMIRNICA **root) {
    if(*root != NULL) {
        destroy_tree(&((*root)->left));
        destroy_tree(&((*root)->right));
        free(*root);
        *root = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

NAMIRNICA *get_najbolja_namirnica(NAMIRNICA *root) {
    if (root == NULL) { // tree is empty
        return NULL;
    }

    // Stablo je uredjeno po kljucu "kolicina", u opadajucem redosledu.
    // Stoga se namirnica sa najvise vitamina C sigurno nalazi u krajnje levom
    // listu.

    if (root->left == NULL) {
        // ovaj "root" je krajnje levi list, posto nema levo dete
        return root;
    }

    // Trazi dalje, *sigurno* ima namirnica sa jos vise vitamina C
    return get_najbolja_namirnica(root->left);
}

int main(int arg_num, char *args[]) {
    if (arg_num != 3) {
        printf("USAGE: %s IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *in_filename = args[1];
    char *out_filename = args[2];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    NAMIRNICA *root;
    init_tree(&root);
```

```
    read_tree_from(in, &root);
    save_tree_to(out, root);

    NAMIRNICA *best = get_najbolja_namirnica(root);
    if (best != NULL) {
        fprintf(out, "\nNamirnica sa najvise vitamina C je:\n");
        save_item_to(out, best);
    }

    destroy_tree(&root);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.12 Zadatak „Ponovo radi bioskop”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati podatke u binarno stablo, gde struktura `projekcija_st` sadrži sledeća polja:

- Skraćena oznaka grada (jedna reč, tačno 2 karaktera)
- Naziv bioskopa (jedna reč, do 10 karaktera)
- Naziv filma (jedna reč, do 8 karaktera)
- Naziv žanra (jedna reč, do 9 karaktera)
- Cena ulaznice (pozitivan realan broj)

Naravno, struktura `projekcija_st` sadrži i polja potrebna za pravilno formiranje binarnog stabla.

Na osnovu zadatog grada `grad` i žanra `zanr` iz formiranog binarnog stabla upisati podatke u zadatu izlaznu datoteku, u sledećem rasporedu polja strukture `projekcija_st`:

- Skraćena oznaka grada (koristiti `"%2s"` format specifikator)
- Naziv bioskopa (koristiti `"%-10s"` format specifikator)
- Naziv filma (koristiti `"%-8s"` format specifikator)
- Naziv žanra (koristiti `"%-9s"` format specifikator)
- Cena ulaznice (zaokružena na 2 decimale, koristiti `"%6.2f"` format specifikator)

i potom u istu izlaznu datoteku upisati informaciju koja je projekcija najpovoljnija za zadati grad i žanr.

Primer poziva:

```
./povoljne_ulaznice NS Animated projekcije.txt analiza.txt
```

sa `grad=NS`, `zanr=Animated` i zadatim ulazom u datoteci `projekcije.txt`:

```
NS Arena     Hobit   Fantasy  500.00
NS Arena     Frozen  Animated 366.60
NS Arena     Smurfs  Animated 399.90
NS Arena     Gravity Sci-Fi   300.00
BG Cineplexx Hobit   Fantasy  550.00
BG Cineplexx Frozen  Animated 426.66
BG Cineplexx Smurfs  Animated 429.99
BG Cineplexx Gravity Sci-Fi   230.00
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
BG Cineplexx  Gravity Sci-Fi   230.00
NS Arena      Gravity Sci-Fi   300.00
NS Arena      Frozen  Animated 366.60
NS Arena      Smurfs  Animated 399.90
BG Cineplexx  Frozen  Animated 426.66
BG Cineplexx  Smurfs  Animated 429.99
NS Arena      Hobit   Fantasy  500.00
BG Cineplexx  Hobit   Fantasy  550.00

Najpovoljnija projekcija za Animated filmove u NS je:
Frozen 366.60
```

Primer poziva kada projekcija sa zadatim kriterijumima ne postoji:

```
./povoljne_ulaznice NS Horor projekcije_za_decu.txt analiza.txt
```

sa `grad=NS`, `zanr=Horor`, zadatim ulazom u datoteci `projekcije_za_decu.txt`:

```
NS Arena     Hobit   Fantasy  500.00
NS 4Kids     Frozen  Animated 366.60
NS 4Kids     Smurfs  Animated 399.90
```

i očekivanim izlazom u datoteci `analiza.txt`:

```
NS 4Kids     Frozen  Animated 366.60
NS 4Kids     Smurfs  Animated 399.90
NS Arena     Hobit   Fantasy  500.00

Niko ne projektuje Horor filmove u NS!
```

4.12.1 Primer rešenja

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV_GRAD 2+1
```

```
#define MAX_NAZIV_BIOSKOP 10+1
#define MAX_NAZIV_FILM 8+1
#define MAX_NAZIV_ZANR 9+1

typedef struct projekcija_st {
    char grad[MAX_NAZIV_GRAD];
    char bioskop[MAX_NAZIV_BIOSKOP];
    char film[MAX_NAZIV_FILM];
    char zanr[MAX_NAZIV_ZANR];
    double cena;
    struct projekcija_st *left;
    struct projekcija_st *right;
} PROJEKCIJA;

void init_tree(PROJEKCIJA **root) {
    *root = NULL;
}

void add_to_tree(PROJEKCIJA *new, PROJEKCIJA **root) {
    if(*root == NULL) { // tree is empty
        *root = new;
    } else if (new->cena <= (*root)->cena) {
        add_to_tree(new, &((*root)->left));
    } else {
        add_to_tree(new, &((*root)->right));
    }
}

PROJEKCIJA *create_new_item(char grad[], char bioskop[], char film[],
char zanr[], double cena) {
    PROJEKCIJA *new = (PROJEKCIJA *)malloc(sizeof(PROJEKCIJA));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->grad, grad);
    strcpy(new->bioskop, bioskop);
    strcpy(new->film, film);
    strcpy(new->zanr, zanr);
    new->cena = cena;

    new->left = NULL;
    new->right = NULL;

    return new;
}

void read_tree_from(FILE *in, PROJEKCIJA **root) {
    char tmp_grad[MAX_NAZIV_GRAD];
    char tmp_bioskop[MAX_NAZIV_BIOSKOP];
    char tmp_film[MAX_NAZIV_FILM];
    char tmp_zanr[MAX_NAZIV_ZANR];
    double tmp_cena;

    while(fscanf(
        in, "%s %s %s %s %lf",
```

```
         tmp_grad,
         tmp_bioskop,
         tmp_film,
         tmp_zanr,
        &tmp_cena
    ) != EOF) {
        PROJEKCIJA *new = create_new_item(
            tmp_grad, tmp_bioskop, tmp_film, tmp_zanr, tmp_cena
        );
        add_to_tree(new, root);
    }
}

void save_item_to(FILE *out, PROJEKCIJA *x) {
    fprintf(
        out, "%2s %-10s %-8s %-9s %6.2f\n",
        x->grad, x->bioskop, x->film, x->zanr, x->cena
    );
}

void save_tree_to(FILE *out, PROJEKCIJA *root) {
    if(root != NULL) {
        save_tree_to(out, root->left);
        save_item_to(out, root);
        save_tree_to(out, root->right);
    }
}

void destroy_tree(PROJEKCIJA **root) {
    if(*root != NULL) {
        destroy_tree(&((*root)->left));
        destroy_tree(&((*root)->right));
        free(*root);
        *root = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

PROJEKCIJA *get_najpovoljnija_projekcija(PROJEKCIJA *root, char grad[],
char zanr[]) {
    if (root == NULL) { // tree is empty
        return NULL;
    }

    PROJEKCIJA *best = NULL;
    if (strcmp(root->grad, grad) == 0 && strcmp(root->zanr, zanr) == 0) {
        // Gledamo samo projekcije koje su OK
        best = root;
    }
```

```
    PROJEKCIJA *left = get_najpovoljnija_projekcija(root->left, grad, zanr);
    if (left != NULL) {
        // Nadjena OK projekcija u levom podstablu
        if (best == NULL || left->cena < best->cena) {
            // Nadjena povoljnija projekcija, koja je OK
            best = left;
        }
    }

    // Posto stablo nije uredjeno po sva 3 kriterijuma, postoji mogucnost da se
    // trazena projekcija nalazi i u desnom podstablu
    PROJEKCIJA *right = get_najpovoljnija_projekcija(root->right, grad, zanr);
    if (right != NULL) {
        // Nadjena OK projekcija u desnom podstablu
        if (best == NULL || right->cena < best->cena) {
            // Nadjena povoljnija projekcija, koja je OK
            best = right;
        }
    }

    return best;
}

int main(int arg_num, char *args[]) {
    if (arg_num != 5) {
        printf("USAGE: %s GRAD ZANR IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *grad = args[1];
    char *zanr = args[2];
    char *in_filename = args[3];
    char *out_filename = args[4];

    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    PROJEKCIJA *root;
    init_tree(&root);

    read_tree_from(in, &root);
    save_tree_to(out, root);

    PROJEKCIJA *best = get_najpovoljnija_projekcija(root, grad, zanr);
    if (best == NULL) {
        fprintf(
            out, "\nNiko ne projektuje %s filmove u %s!\n",
            zanr, grad
        );
    } else {
        fprintf(
            out, "\nNajpovoljnija projekcija za %s filmove u %s je:\n%s %.2f\n",
            zanr, grad, best->film, best->cena
        );
    }
```

```
    destroy_tree(&root);

    fclose(in);
    fclose(out);

    return 0;
}
```

4.13 Zadatak „Udaljenost planeta”

Autor zadatka: Petar Marić <petarmaric@uns.ac.rs>

Iz zadate ulazne datoteke učitati podatke u jednostruko spregnutu listu, gde struktura `planeta_st` sadrži sledeća polja:

- Naziv planete (jedna reč, do 16 karaktera)
- X koordinata planete (ceo broj)
- Y koordinata planete (ceo broj)
- Z koordinata planete (ceo broj)

Naravno, struktura `planeta_st` sadrži i polja potrebna za pravilno formiranje jednostruko spregnute liste.

Pronaći 2 planete čija je međusobna udaljenost najveća. U zadatu izlaznu datoteku upisati 3 linije:

- Informacije o prvoj planeti
- Informacije o drugoj planeti
- Udaljenost između 2 planete (zaokružena na 2 decimale, koristiti `"%.2f"` format specifikator)

Smatrati da će u ulaznoj datoteci uvek biti navedene najmanje 2 planete.

Primer poziva:

```
./udaljenost planete.txt izvestaj.txt
```

sa zadatim ulazom u datoteci `planete.txt`:

```
Merkur  20 100 3
Zemlja -29 180 5
Mars   124 270 6
Saturn  20 350 9
```

i očekivanim izlazom u datoteci `izvestaj.txt`:

```
Merkur 20 100 3
Saturn 20 350 9
250.07
```

4.13.1 Primer rešenja

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAZIV 16+1

typedef struct planeta_st {
    char naziv[MAX_NAZIV];
    int x;
    int y;
    int z;
    struct planeta_st *next;
} PLANETA;

void init_list(PLANETA **head) {
    *head = NULL;
}

void add_to_list(PLANETA *new, PLANETA **head) {
    if(*head == NULL) { // list is empty
        *head = new;
        return;
    }

    add_to_list(new, &((*head)->next));
}

PLANETA *create_new_item(char naziv[], int x, int y, int z) {
    PLANETA *new = (PLANETA *)malloc(sizeof(PLANETA));
    if (new == NULL) {
        printf("Not enough RAM!\n");
        exit(21);
    }

    strcpy(new->naziv, naziv);
    new->x = x;
    new->y = y;
    new->z = z;

    new->next = NULL;

    return new;
}

void read_list_from(FILE *in, PLANETA **head) {
    char tmp_naziv[MAX_NAZIV];
    int tmp_x;
    int tmp_y;
    int tmp_z;

    while(fscanf(in, "%s %d %d %d", tmp_naziv, &tmp_x, &tmp_y, &tmp_z) != EOF) {
        PLANETA *new = create_new_item(tmp_naziv, tmp_x, tmp_y, tmp_z);
        add_to_list(new, head);
    }
}
```

```
void save_item_to(FILE *out, PLANETA *x) {
    fprintf(
        out, "%s %d %d %d\n",
        x->naziv, x->x, x->y, x->z
    );
}

void destroy_list(PLANETA **head) {
    if(*head != NULL) {
        destroy_list(&((*head)->next));
        free(*head);
        *head = NULL;
    }
}

FILE *safe_fopen(char *filename, char *mode, int error_code) {
    FILE *fp = fopen(filename, mode);
    if (fp == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(error_code);
    }
    return fp;
}

double udaljenost(PLANETA *a, PLANETA *b) {
    return sqrt(
        pow(a->x - b->x, 2) +
        pow(a->y - b->y, 2) +
        pow(a->z - b->z, 2)
    );
}

void find_max_udaljenost(PLANETA *head, PLANETA **p1, PLANETA **p2) {
    *p1 = head;
    *p2 = head->next;

    PLANETA *a;
    PLANETA *b;
    for(a = head; a != NULL; a = a->next) {
        for(b = a->next; b != NULL; b = b->next) {
            if (udaljenost(a, b) > udaljenost(*p1, *p2)) {
                *p1 = a;
                *p2 = b;
            }
        }
    }
}

int main(int arg_num, char *args[]) {
    if (arg_num != 3) {
        printf("USAGE: %s IN_FILENAME OUT_FILENAME\n", args[0]);
        exit(11);
    }

    char *in_filename = args[1];
    char *out_filename = args[2];
```

```
    FILE *in  = safe_fopen(in_filename,  "r", 1);
    FILE *out = safe_fopen(out_filename, "w", 2);

    PLANETA *head;
    init_list(&head);

    read_list_from(in, &head);

    PLANETA *p1;
    PLANETA *p2;
    find_max_udaljenost(head, &p1, &p2);
    save_item_to(out, p1);
    save_item_to(out, p2);
    fprintf(out, "%.2f\n", udaljenost(p1, p2));

    destroy_list(&head);

    fclose(in);
    fclose(out);

    return 0;
}
```

CHAPTER

5

LICENCA

informational purposes only, are not exhaustive, and do not form part of our licenses.

> Considerations for licensors: Our public licenses are intended for use by those authorized to give the public permission to use material in ways otherwise restricted by copyright and certain other rights. Our licenses are irrevocable. Licensors should read and understand the terms and conditions of the license they choose before applying it. Licensors should also secure all rights necessary before applying our licenses so that the public can reuse the material as expected. Licensors should clearly mark any material not subject to the license. This includes other CC- licensed material, or material used under an exception or limitation to copyright. More considerations for licensors:
>
> wiki.creativecommons.org/Considerations_for_licensors
>
> Considerations for the public: By using one of our public licenses, a licensor grants the public permission to use the licensed material under specified terms and conditions. If the licensor's permission is not necessary for any reason–for example, because of any applicable exception or limitation to copyright–then that use is not regulated by the license. Our licenses grant only permissions under copyright and certain other rights that a licensor has authority to grant. Use of the licensed material may still be restricted for other reasons, including because others have copyright or other rights in the material. A licensor may make special requests, such as asking that all changes be marked or described. Although not required by our licenses, you are encouraged to respect those requests where reasonable. More considerations for the public:
>
> wiki.creativecommons.org/Considerations_for_licensees

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License („Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. Adapted Material means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which

the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. Adapter's License means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. BY-NC-SA Compatible License means a license listed at creativecommons.org/compatiblelicenses, approved by Creative Commons as essentially the equivalent of this Public License.

d. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

e. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

f. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

g. License Elements means the license attributes listed in the name of a Creative Commons Public License. The License Elements of this Public License are Attribution, NonCommercial, and ShareAlike.

h. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

i. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

j. Licensor means the individual(s) or entity(ies) granting rights under this Public License.

k. NonCommercial means not primarily intended for or directed towards commercial advantage or monetary compensation. For purposes of this Public License, the exchange of the Licensed Material for other material subject to Copyright and Similar Rights by digital file-sharing or similar means is NonCommercial provided there is no payment of monetary compensation in connection with the exchange.

l. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

m. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

n. You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

a. reproduce and Share the Licensed Material, in whole or in part, for NonCommercial purposes only; and

b. produce, reproduce, and Share Adapted Material for NonCommercial purposes only.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a) (4) never produces Adapted Material.

5. Downstream recipients.

a. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor

to exercise the Licensed Rights under the terms and conditions of this Public License.

b. Additional offer from the Licensor – Adapted Material. Every recipient of Adapted Material from You automatically receives an offer from the Licensor to exercise the Licensed Rights in the Adapted Material under the conditions of the Adapter's License You apply.

c. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties, including when the Licensed Material is used other than for NonCommercial purposes.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

a. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

b. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

c. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

b. ShareAlike.

In addition to the conditions in Section 3(a), if You Share Adapted Material You produce, the following conditions also apply.

1. The Adapter's License You apply must be a Creative Commons license with the same License Elements, this version or later, or a BY-NC-SA Compatible License.

2. You must include the text of, or the URI or hyperlink to, the Adapter's License You apply. You may satisfy this condition in any reasonable manner based on the medium, means, and context in which You Share Adapted Material.

3. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, Adapted Material that restrict exercise of the rights granted under the Adapter's License You apply.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database for NonCommercial purposes only;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material, including for purposes of Section 3(b); and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION, NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES, COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights

licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

   2. upon express reinstatement by the Licensor.

   For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes

INDEKS ZADATAKA PO KATEGORIJAMA