

## Laboratorijska Vježba 2. Uvod u C# i .NET Platformu

### Cilj vježbe:

- Upoznavanje sa *Visual Studio* okruženjem za razvoj .NET aplikacija;
- Upoznavanje sa načinom kreiranja .NET Core konzolnih aplikacija;
- Učenje osnova programiranja u C# jeziku.

### 1. *Visual Studio* okruženje

Najpoznatije i najčešće korišteno okruženje za rad u programskom jeziku C# je *Visual Studio* okruženje. Ovo okruženje razvijeno je za operativne sisteme *Windows* i *Mac*, dok za operativni sistem *Linux* postoje različita zamjenska okruženja (od kojih je najpopularnije *Rider*, koji je moguće instalirati putem sljedećeg linka: <https://www.jetbrains.com/rider/>). Na laboratorijskim vježbama biti će korišteno **isključivo** *Visual Studio* okruženje, ali to ne predstavlja nikakvu prepreku za rad u timovima gdje studenti koriste različite operativne sisteme, jer je platforma koja će se koristiti **.NET Core platforma**, koja omogućava rad u različitim operativnim sistemima bez ikakvih ograničenja. To zapravo znači da, uz eventualne male izmjene (do čega dolazi pri korištenju eksternih nastavaka koji u zamjenskim okruženjima nemaju ista imena i sl.), isti projekat sa programskim kodom može biti pokrenut u *Visual Studio* i *Rider* okruženju bez da dolazi i do kakvog konflikta i nemogućnosti pokretanja aplikacije, što ovu platformu čini osnovom za rad u programskom jeziku C#.

Verzija *Visual Studio* okruženja koja će se koristiti na laboratorijskim vježbama je **Visual Studio 2019 Community**. *Community* je jedina besplatna verzija ovog okruženja koja unosi određena ograničenja u samo okruženje, ali ta ograničenja ne odnose se na funkcionalnosti koje će se koristiti na predmetu. Ovo okruženje moguće je preuzeti na sljedećem linku: <https://visualstudio.microsoft.com/downloads/>. Dozvoljeno je korištenje i starijih verzija okruženja (npr. 2017 ili 2015), ali je potrebno voditi računa da neke od funkcionalnosti u starijim verzijama nemaju isti oblik, kao i da je moguć nastanak velikog broja konflikta ukoliko članovi tima koriste različite verzije okruženja. Važno je napomenuti da je za instalaciju *Visual Studio 2019* okruženja potrebno prethodno izvršiti *update Windows* operativnog sistema, jer u suprotnom instalacija okruženja neće biti dozvoljena.

Pri instalaciji okruženja potrebno je odabrati pojedinačne funkcionalnosti koje se žele koristiti, odnosno različite dijelove *Visual Studio* okruženja koje je potrebno instalirati kako bi se mogao vršiti razvoj različitih vrsta aplikacija. Za rad na ovom predmetu potrebno je instalirati sljedeće funkcionalnosti (kao što je prikazano na Slici 1):

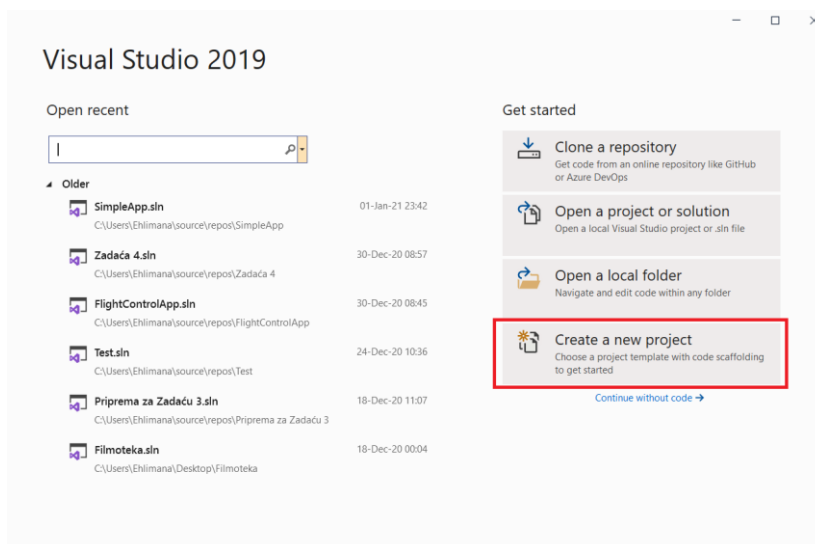
1. *ASP.NET and Web Development*;
2. *.NET Desktop Development*;
3. *.NET Core Cross-Platform Development*.



Slika 1. Odabir funkcionalnosti Visual Studio okruženja

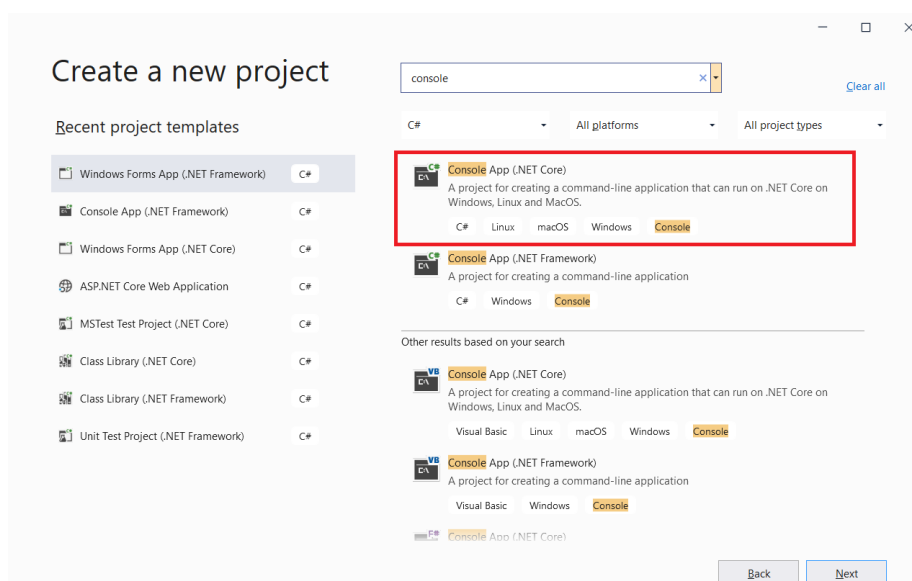
## 2. Kreiranje prve konzolne aplikacije

Nakon instalacije *Visual Studio* okruženja, mogu se početi praviti C# aplikacije. U ovoj laboratorijskoj vježbi kreirati će se prva konzolna aplikacija kako bi se na jednostavan način počelo koristiti okruženje i demonstrirali osnovni koncepti C# programskog jezika. Prvo je potrebno pokrenuti okruženje, nakon čega će korisniku biti ponuđeno da otvori neku postojeću aplikaciju (lokalno ili u okviru dijeljenog repozitorija) ili da kreira novi projekat. Potrebno je odabrati opciju **Create a new project**, kao što je prikazano na Slici 2.



Slika 2. Početni ekran Visual Studio okruženja

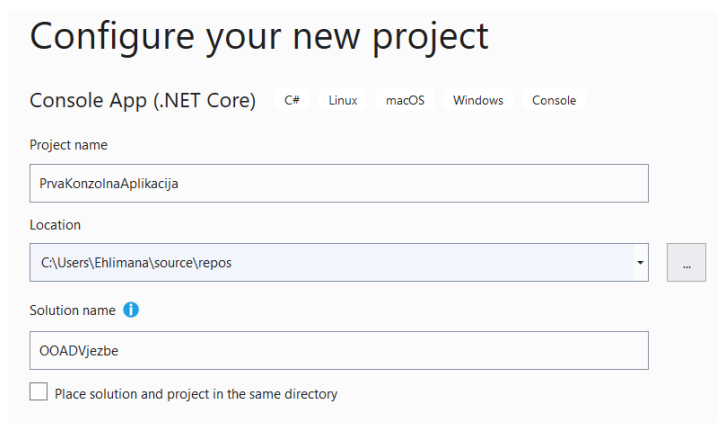
Nakon odabira opcije za kreiranje novog projekta, korisniku se prikazuju različite vrste aplikacija koje su podržane okruženjem. Koje aplikacije je moguće kreirati ovisi od instaliranih funkcionalnosti *Visual Studio* okruženja. U okviru instalirane funkcionalnosti **.NET Core Cross-Platform Development** nalaze se i konzolne aplikacije prilagođene za sve operativne sisteme, pa je potrebno odabrati tu vrstu aplikacije. Najlakše je izvršiti pretragu unosom pojma *console*, na način prikazan na Slici 3. Kao što je vidljivo na slici, postoji više vrsta ponuđenih konzolnih aplikacija – konzolne aplikacije koje je moguće pokretati samo na *Windows* operativnom sistemu (*.NET Framework* vrsta), kao i konzolne aplikacije koje nemaju oznaku **C#** nego **VB**, što označava da se takve aplikacije ne programiraju u C#, nego u Visual Basic programskom jeziku. Potrebno je voditi računa da se odabere ispravan tip aplikacije, kao i ispravan programski jezik u kojem će se vršiti razvoj.



Slika 3. Odabir željene vrste konzolne aplikacije

Nakon vršenja odabira željene vrste aplikacije, potrebno je izvršiti još konfiguraciju osnovnih informacija o aplikaciji, kao što je prikazano na Slici 4. Osim lokacije na koju će se aplikacija sačuvati (najčešće je to predefinisani direktorij za sve aplikacije kreirane u *Visual Studio* okruženju), potrebno je odabrati ime **projekta** (*project*) i **programskog rješenja** (*solution*). Programsko rješenje apstraktniji je pojam od projekta, jer jedno programsko rješenje može sadržati više projekata (npr. projekat za desktop aplikaciju, projekat za web-aplikaciju, biblioteku klasa i testove za ostale projekte).

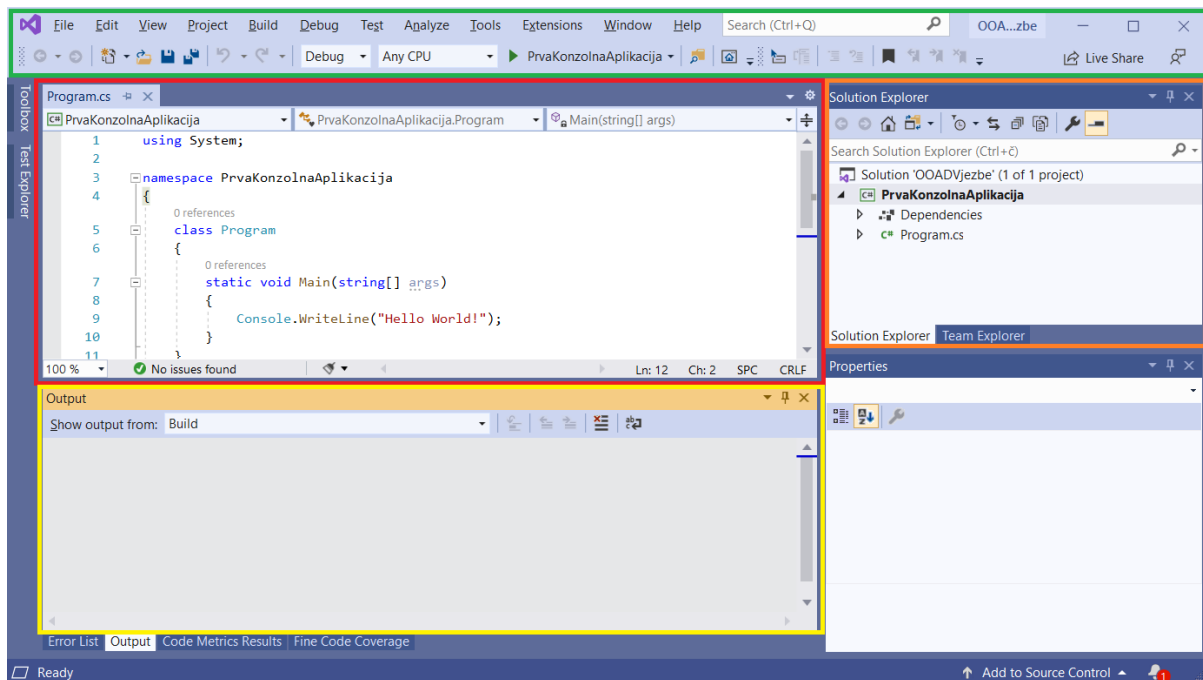
U ovoj laboratorijskoj vježbi kreirati će se samo jedan projekat koji je prethodno odabran (*.NET Core* konzolna aplikacija), tako da se i programskom rješenju i projektu može dati isto ime – *PrvaKonzolnaAplikacija*, dok je u slučaju kompleksnijih projekata preporučeno korištenje različitog imenovanja za programsko rješenje i sve pojedinačne projekte kako ne bi došlo do zabune o namjeni svakog pojedinačnog projekta u odnosu na cijelo programsko rješenje. Postoji i opcija za pozicioniranje projekta i programskog rješenja u isti direktorij, što je preporučeno učiniti samo u slučaju da će se programsko rješenje sastojati samo od jednog projekta.



Slika 4. Dodavanje osnovnih informacija o aplikaciji

Nakon kreiranja novog programskog rješenja, korisnik dobiva pristup samom okruženju, koje je prikazano na Slici 5. Okruženje se sastoji od četiri najvažnije cjeline koje su na slici označene različitim bojama i koje će biti opisane u nastavku. Okruženje omogućava pristup i dodatnim funkcionalnostima o kojima neće biti riječi u okviru ove laboratorijske vježbe (npr. *Properties* i *Toolbox* tabovi), ali koje će biti korištene pri razvoju drugih vrsta aplikacija.

- *Menu trake* (označene zelenom bojom) – ove trake omogućavaju pristup svim funkcionalnostima okruženja. U okviru njih moguće je kreirati novi ili otvoriti postojeći projekat, promijeniti prikaz tako da se u okruženje dodaje neki tab kojem se prethodno nije moglo pristupiti, promijeniti postavke i sl. Najvažniji dio ovog dijela okruženja je *Start* dugme (zeleni trokutić pored kojeg piše ime aplikacije *PrvaKonzolnaAplikacija*) koje omogućava pokretanje aplikacije, ukoliko je to moguće.
- *Glavni prozor* (označen crvenom bojom) – u ovom dijelu okruženja prikazuju se svi otvoreni *file*-ovi. Okruženje dozvoljava otvaranje C# *file*-ova (koji imaju nastavak CS), ali i drugih vrsta *file*-ova kojima se definišu postavke (npr. CSProj), izvori podataka (npr. TXT, CSV ili XML) ili koji koriste *markup* sintaksu (npr. CSHTML ili CSS). U ovom prozoru vrši se editovanje programskog koda.
- *Solution explorer* (označen narandžastom bojom) – ovaj dio okruženja koristi se za pregled strukture foldera od kojih se programsko rješenje sastoji, kao i za dodavanje, brisanje i izmjenu *file*-ova. **Dependencies** oznaka omogućava pristup bibliotekama klasa i ekstenzijama koje se koriste u programskom rješenju.
- *Izlazni prozori* (označeni žutom bojom) – u ovom dijelu okruženja prikazuju se dva veoma važna taba *Output* i *Error List* koji daju izlazne informacije nakon pokretanja aplikacije. Ukoliko se aplikacija ne može pokrenuti, moguće je pregledati sve greške koje je potrebno ispraviti, a pri pokretanju se dobivaju informacije o uspješnosti i eventualne dodatne informacije koje mogu biti važne za korisnika. U ovom dijelu okruženja prikazuje se i *Nuget Package Manager Console* (NPM) tab koji služi za instalaciju ekstenzija, o čemu će biti riječi u nekoj od narednih laboratorijskih vježbi.



Slika 5. Prikaz Visual Studio okruženja

Sada se može početi sa razvojem prve konzolne aplikacije koristeći programski jezik C#, što će biti detaljno objašnjeno u nastavku.

### 3. Korištenje programskog jezika C# za kreiranje konzolnih aplikacija

Korištenje osnovnih principa programskog jezika C# za kreiranje prve konzolne aplikacije biti će demonstrirano koristeći jednostavni primjer. Neka je potrebno kreirati aplikaciju koja od korisnika traži unos šest cijelih brojeva, a zatim ispisuje da li su svi uneseni brojevi pozitivni i ispisuje sve neparne brojeve, ukoliko ih ima.

Ovu aplikaciju moguće je razdvojiti na tri dijela:

- Unos podataka i njihovo pretvaranje u odgovarajući format;
- Provjera da li su svi brojevi pozitivni i pronalazak neparnih brojeva, ukoliko ih ima;
- Prikaz rezultata obrade korisniku.

Prvo se treba u konzoli ispisati poruka korisniku da se od njega očekuje unos brojeva, a zatim omogućiti korisniku unos. U tu svrhu koristi se predefinisana klasa **Console** i njene metode **WriteLine** i **ReadLine**. Nakon toga potrebno je unesene podatke pretvoriti u cijele brojeve koji će se koristiti u daljem dijelu zadatka. Kako metoda **Console.ReadLine** učitava korisnički unos u **string** varijablu, potrebno je prvo razdvojiti cijeli korisnički unos (npr. „1,2,3,4,5,6“) u pojedinačne **string** varijable, a zatim pretvoriti zasebne stringove u cijele brojeve. Cijeli korisnički unos razdvaja se u zasebne **string** varijable koristeći metodu **Split** klase **string**, a pretvaranje u cijele brojeve vrši se koristeći klasu **Int32** i njenu metodu **Parse**. Postoji i metoda **TryParse** koja se može koristiti ukoliko se želi izbjeći pojava izuzetaka (a i cijeli dio koda se jednostavno može staviti u *try-catch* blok ukoliko je neophodno).

```
using System;

namespace PrvaKonzolnaAplikacija
{
    class Program
    {
        static void Main(string[] args)
        {
            // unos brojeva
            Console.WriteLine("Unesite šest prirodnih brojeva (razdvojenih
zarezm):");
            string ulaz = Console.ReadLine();

            // razdvajanje unosa na pojedinačne brojeve
            string[] pojedinačniUlazi = ulaz.Split(",");

            // pretvaranje string brojeva u cijele brojeve (int)
            int[] brojevi = new int[6];
            for (int i = 0; i < 6; i++)
                brojevi[i] = Int32.Parse(pojedinačniUlazi[i]);
        }
    }
}
```

Kao što je vidljivo u listingu, u aplikaciji se koristi samo osnovna biblioteka *System* koja omogućava rad sa osnovnim tipovima programskog jezika C#. *Namespace* definiše opseg u kojem su definisane klase vidljive, o čemu je važno voditi računa pri definisanju više projekata. Konzolne aplikacije imaju predefinisanu klasu **Program** sa metodom **Main** koja se izvršava pri pokretanju programa. Zbog uštede prostora, u narednim isječcima koda neće biti prikazan ovaj dio koda.

Provjera da li su svi brojevi pozitivni i pronalazak neparnih brojeva može se jednostavno napraviti koristeći osnovne principe programiranja. Kako se u ovom zadatku koriste klasični nizovi, dovoljno je koristiti *for* petlju za prolazak kroz sve elemente niza, ali je mnogo lakše i više intuitivno korištenje *foreach* petlje. Ova petlja omogućava korisniku da, umjesto korištenja brojačke varijable, pristupa privremenim varijablama koje označavaju elemente niza.

```
// obrada podataka
bool pozitivni = true;
int[] neparni = new int[6];
int brojNeparnih = 0;

// prolazak kroz sve elemente niza
foreach(var element in brojevi)
{
    if (element <= 0)
        pozitivni = false;

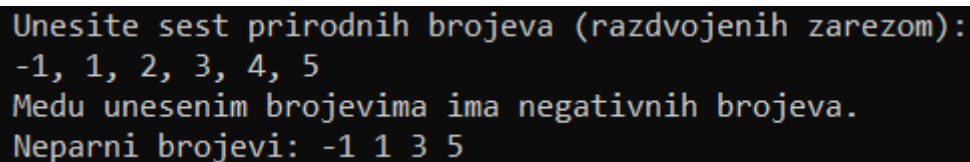
    if (element % 2 != 0)
    {
        neparni[brojNeparnih] = element;
        brojNeparnih++;
    }
}
```



Sada je neophodno ispisati rezultate – da li su svi brojevi koje je korisnik unio pozitivni, kao i sve neparne brojeve koje unos sadrži. To se vrši koristeći metodu *WriteLine* klase *Console*, pri čemu se kao olakšice koriste **ternarni operator** (koji omogućava pojednostavljenje *if-else* logike koristeći operatore *? i :*) i metodu *string.Join* (koja zamjenjuje *for* petlju za ispisivanje svih pojedinačnih elemenata niza koristeći brojačku varijablu). Potrebno je prikazati samo neparne elemente (kojih uvijek ima manje ili isto koliko i unesenih brojeva), pa je iz tog razloga potrebno za prikaz odabrati samo one elemente koji su dodani u niz neparanih brojeva (što se vrši koristeći indeksiranje koristeći početni i krajnji element, sintaksom **niz[prvi..posljednji]**).

```
// prikaz rezultata
string prikaz = pozitivni == true ? "nema" : "ima";
Console.WriteLine(" Među unesenim brojevima " + prikaz + " negativnih
brojeva.");
Console.WriteLine(" Neparni brojevi: " + string.Join(" ",
neparni[0..brojNeparnih]));
```

Izgled korisničkog unosa i izlaza iz aplikacije nakon obrade podataka prikazan je na Slici 6. Korisnik smije unijeti brojeve sa proizvoljnim brojem blanko znakova (koji se ignorišu pri pretvaranju u cijele brojeve), samo je važno da se brojevi razdvoje zarezom. Kako unos sadrži broj **-1**, u konzoli se ispisuje da u nizu ima negativnih brojeva, a zatim se ispisuju svi neparni brojevi (uključujući **-1**, koji bez obzira na to što je negativan, kao rezultat pri dijeljenju s 2 ne daje ostatak 0).



```
Unesite sest prirodnih brojeva (razdvojenih zarezom):
-1, 1, 2, 3, 4, 5
Među unesenim brojevima ima negativnih brojeva.
Neparni brojevi: -1 1 3 5
```

Slika 6. Izlaz definisane aplikacije u konzoli

Na prvi pogled prethodno kreirana aplikacija čini se jednostavnom i lakom za korištenje, ali stil programiranja koji koristi nizove i alokaciju ne koristi se često u programskom jeziku C# i svojstven je nižim programskim jezicima poput C++. U C# programskom jeziku postoji veliki broj predefinisanih biblioteka koje olakšavaju rad sa kolekcijama podataka i mnogim drugim vrstama podataka, što će biti demonstrirano u sljedećoj laboratorijskoj vježbi.

#### 4. Zadaci za samostalni rad

Programski kod aplikacije koja je kreirana u vježbi dostupan je u repozitoriju na sljedećem linku: <https://github.com/ehlymana/OOADVjezbe>, na *branchu* **lv2**.

1. Napišite program koji traži da se sa tastature unese brzina broda u čvorovima koja se zadaje isključivo kao cijeli broj (obavezno koristiti varijablu tipa *int*), a zatim izračunava i ispisuje brzinu broda u km/h kao decimalan broj. Koristite činjenicu da je čvor morska milja na sat, a da je jedna morska milja 1852 m. Naprimjer, ukoliko se kao brzina broda unese broj 20, program treba da ispiše rezultat 37.04 jer je 20 čvorova = 37.04 km/h.
2. Napišite program koji traži da se sa tastature unese cijeli broj *n*, a zatim isertava na ekranu jednakostranični trougao sastavljen od zvjezdica čija je osnovica horizontalna a vrh usmjeren nagore. Naprimjer, ukoliko se unese  $n = 4$ , ispis na ekranu treba da izgleda na sljedeći način:

```
      *
     ***
    *****
   ********
```

3. Napišite program koji traži da se sa tastature unese prirodan broj *n*, a nakon toga se unose elementi kvadratne matrice formata  $n \times n$ . Program nakon toga treba da ispiše redni broj kolone sa najvećom sumom elemenata, redni broj reda sa najmanjom sumom elemenata, kao i sumu elemenata na dijagonali.
4. Napišite funkciju “DaLiJePalindrom” koja za string koji joj je proslijeđen kao parametar ispituje da li predstavlja palindrom ili nije, i kao rezultat vraća odgovarajuću logičku vrijednost *true* ili *false*. Palindromima se smatraju riječi ili rečenice koje se isto čitaju sa obje strane (“kapak”). Prilikom ispitivanja treba ignorisati razmake, interpunkcijske znakove i razliku između velikih i malih slova, tako da rečenica “Ana voli Milovana” treba da bude prepoznata kao palindrom, iako bukvalno pročitana sa suprotnog kraja glasi “anavoliM ilov anA”.
5. Napišite program koji traži od korisnika da unese spisak riječi (broj riječi se prethodno unosi sa tastature), a zatim ispisuje na ekran prvu i posljednju riječ iz spiska po abecednom poretku, kao i popis svih unesenih riječi, ali bez ispisivanja duplikata (tj. bez ispisivanja riječi koje su se već jednom ispisale). Program realizirajte korištenjem niza stringova.