Uvod u programiranje

- predavanja -

prosinac 2025.

19. Dinamička alokacija memorije

Motivacija

- Primjer:
  - S tipkovnice učitavati realne brojeve dok se ne učita broj 999.
    Ispisati brojeve u obrnutom poretku od onog u kojem su učitani.
  - primjeri izvršavanja programa

```
Upisite 1. broj > 9.1↵
Upisite 2. broj> 999↵

999.0, 9.1↵
```

```
Upisite 1. broj > 28.6↵
Upisite 2. broj> 71.425↵
…
Upisite 10000. broj > 999↵

999.0, …, 71.4, 28.6↵
```

- Kako tijekom pisanja programa odrediti ispravan broj članova polja u koje ćemo spremiti vrijednosti učitane s tipkovnice?
- Za prvi primjer izvršavanja dovoljno je polje od 2 člana, a za drugi treba polje od 10000 članova.

Alokacija memorije

- Tijekom pisanja i prevođenja programa ponekad nisu poznate točne potrebe za memorijom.
- U tom slučaju odluku o alokaciji memorije treba donijeti (dinamički) tijekom izvršavanja programa.
- Memoriju za varijable je moguće alocirati:
  - Statički
    Veličina i vrsta memorije mora biti poznata za vrijeme prevođenja.
    - automatske ili registarske varijable na stogu
    - globalne i statičke varijable u posebnom memorijskom području
  - Dinamički
    Alocira se tijekom izvršavanja jer veličina nije poznata za vrijeme prevođenja.
    - na stogu (VLA polja)
    - na gomili

Stog i gomila

- Stog i gomila (*heap*) su odvojena područja u memoriji koja se koriste za različite svrhe.
- Tipična veličina stoga je 1 MB do 8 MB
- Tipična veličina gomile je nekoliko GB
- Prekomjerna upotreba stoga (npr. zbog previše rekurzivnih poziva funkcije) ili gomile (zbog alokacije prevelike količine memorije) može dovesti do grešaka poput 'stack overflow' ili 'out of memory'."

Dinamička alokacija memorije <stdlib.h>

```
void *malloc(size_t size);
```

- pronalazi u memoriji size slobodnih bajtova i vraća pokazivač na njih; ako nema dovoljno slobodne memorije, vraća NULL

```
void *calloc(size_t num, size_t size);
```

- pronalazi u memoriji mjesto za num objekata pojedinačne veličine size, inicijalizira ih na 0 i vraća pokazivač na njih; ako nema dovoljno slobodne memorije, vraća NULL

```
void *realloc(void *ptr, size_t size);
```

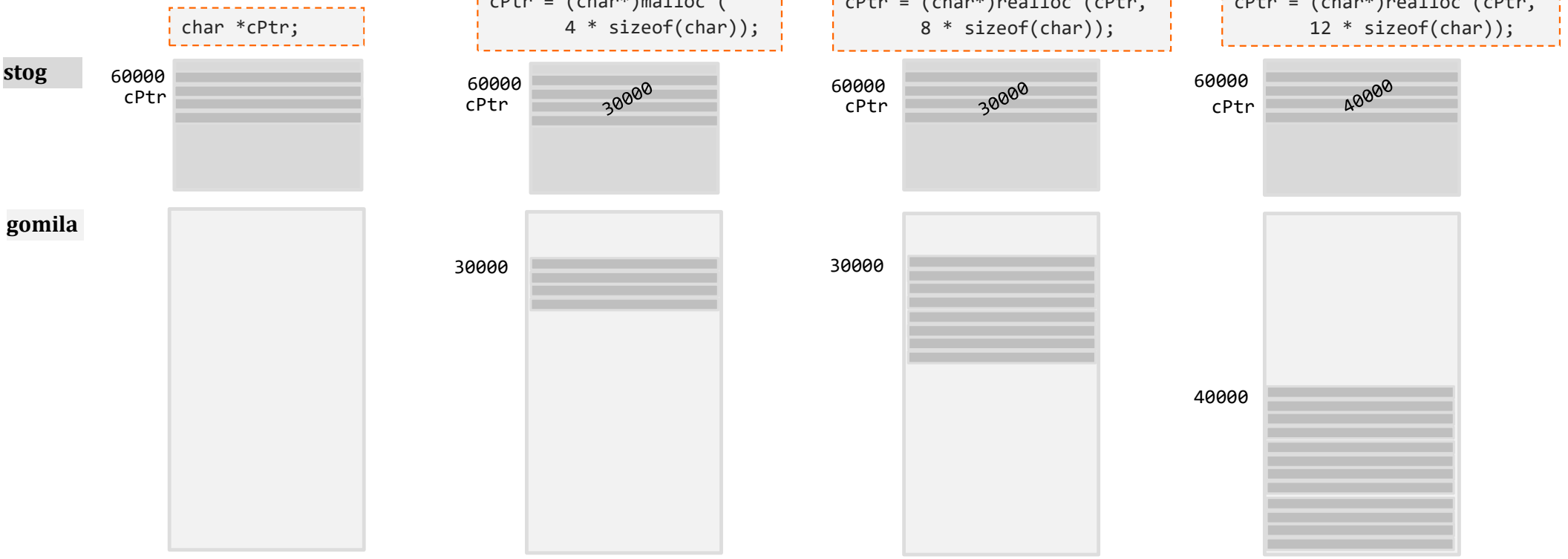
- pokušava promijeniti (povećati/smanjiti) alokaciju na koju pokazuje pokazivač ptr na novu veličinu size; vraća pokazivač na promijenjenu alokaciju;
- ako nema dovoljno slobodne memorije, vraća NULL pri čemu originalni blok memorije ostaje nepromijenjen, tj. ne oslobađa se niti premješta

```
void free(void *ptr);
```

- oslobađa memoriju na koju pokazuje ptr;
- pri tom se ptr ne postavlja na NULL pokazivač pa je to dobro učiniti programski

Dinamička alokacija memorije

```
char *cPtr;
cPtr = (char *)malloc(4 * sizeof(char));
cPtr = (char *)realloc(cPtr, 8 * sizeof(char));
cPtr = (char *)realloc(cPtr, 12 * sizeof(char));
```



Kako rade malloc i realloc

```
char *cPtr;
cPtr = (char *)malloc(4 * sizeof(char));
cPtr = (char *)realloc(cPtr, 8 * sizeof(char));
cPtr = (char *)realloc(cPtr, 12 * sizeof(char));
```

- cPtr je automatska varijabla funkcije i memorija za nju dodjeljuje se **na stogu** (adresa početnog bytea je 60000)
- Naredbom

  `cPtr = malloc(4 * sizeof(char));`

  alocira se 4 bytea memorije **na gomili** (adresa početnog bytea 30000), a cPtr sadrži adresu prvog bajta memorije alocirane na gomili
- Naredbama

  `cPtr = realloc(p, 8 * sizeof(char));`

  `cPtr = realloc(p, 12 * sizeof(char));`

  traži se povećanje alokacije memorije i ako se dodatna memorija pronađe, moguće su dvije situacije:
  - Iza bloka na koji pokazuje cPtr memorija **je slobodna** (slučaj kod prve realloc naredbe - adresa početnog bytea je i dalje 30000), alokacija se jednostavno proširuje i realloc vraća istu vrijednost pokazivača kakva je bila prije.
  - Iza bloka na koji pokazuje cPtr memorija **nije slobodna** jer je nastala neka druga alokacija (slučaj kod druge realloc naredbe - adresa početnog bytea je 40000):
    Alocirat će se memorija na prvoj sljedećoj poziciji na kojoj ima 8 (odnosno 12) slobodnih bajta, svi podaci s originalnih memorijskih lokacija kopirat će se na nove lokacije, oslobodit će se stare lokacije i vratiti pokazivač na početak novog bloka memorije.

realloc – što kad nema dodatne memorije?

```
...
cPtr = (char *)realloc(cPtr, 8 * sizeof(char));
cPtr = (char *)realloc(cPtr, 12 * sizeof(char));
```

- Ako realloc naredba vrati NULL jer nema dovoljno dodatne memorije
  - Varijabla cPtr će biti postavljena na NULL pokazivač, ali do tada zauzeta memorija na gomili neće biti oslobođena
  - Izgubili smo pokazivač na tu memoriju i više je ne možemo ni koristiti ni osloboditi
- Zbog toga realloc treba pažljivo koristiti:

```
char *novicPtr = realloc(cPtr, 8 * sizeof(char));
if (novicPtr == NULL) {
   printf("Greška u alokaciji memorije.\n");
   free(cPtr);
   //cPtr = NULL;   ovdje nema previše smisla jer ionako završavamo s programom
   return EXIT_FAILURE;
}
cPtr = novicPtr;
...
```

Rješenje motivacijskog primjera

S tipkovnice učitavati realne brojeve dok se ne učita broj 999.

Ispisati brojeve u obrnutom poretku od onog u kojem su učitani.

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

int main(void) {
   float broj, *brojevi = NULL; //Pokazivac na memoriju u koju ce biti smjesteni brojevi
   float *noviBrojevi = NULL;
   int brojElemenata = 0;

   do {
      printf("Upisite %d. broj > ", brojElemenata+1);
      scanf("%f", &broj);
      if (brojElemenata == 0){
         brojevi =  malloc((brojElemenata + 1) * sizeof(float)); // alokacija memorije za prvi broj
         if (brojevi == NULL){
            printf("Greska u alokaciji memorije.\n");
            return 1; //zavrsetak s greskom
         }
      }
```

Rješenje motivacijskog primjera

```
      else {
         noviBrojevi = realloc(brojevi, (brojElemenata+1) * sizeof(float)); // Dodatna memorija
         if (noviBrojevi == NULL){
            printf("Greska u alokaciji memorije.\n");
            free(brojevi);
            return 1; //zavrsetak s greskom
         }
         brojevi = noviBrojevi;
      }
      *(brojevi + brojElemenata) = broj; // dodavanje unesenog broja u niz
      brojElemenata++;
   } while (broj != 999);
   printf("Brojevi u obrnutom redoslijedu:\n");
   for (int i = brojElemenata - 1; i >= 0; i--)
      printf("%.2f\n", *(brojevi + i));

   free(brojevi); // Oslobađanje memorije
   return 0;
}
```

Komentar rješenja 1

Uokvireno se može izostaviti- realloc se može pozvati bez prethodnog poziva malloc.

Pri prvom pozivu funkcije realloc parametar *brojevi* će imati vrijednost NULL.

```
...
    if (brojElemenata == 0){
        brojevi =  malloc((brojElemenata + 1) * sizeof(float)); // alokacija memorije za prvi broj
        if (brojevi == NULL){
           printf("Greska u alokaciji memorije.\n");
           return 1;
        }
     }
     else {
        noviBrojevi = realloc(brojevi, (brojElemenata+1) * sizeof(float)); // Dodatna memorija
        if (noviBrojevi == NULL){
           printf("Greska u alokaciji memorije.\n");
           free(brojevi);
           return 1;
        }
        brojevi = noviBrojevi;
     }
     …
```

Modificirano rješenje motivacijskog primjera (1)

```
...
int main(void) {
   float broj, *brojevi = NULL, *noviBrojevi = NULL;
   int brojElemenata = 0;

   do {
      printf("Upisite %d. broj > ", brojElemenata+1);
      scanf("%f", &broj);
      noviBrojevi = realloc(brojevi, (brojElemenata+1) * sizeof(float));
      if (noviBrojevi == NULL) {
         printf("Greska u alokaciji memorije.\n");
         free(brojevi);
         return 1; //zavrsetak s greskom
      }
      brojevi = noviBrojevi;
      *(brojevi + brojElemenata) = broj; // dodavanje unesenog broja u niz
      brojElemenata++;
   } while (broj != 999);
   printf("Brojevi u obrnutom redoslijedu:\n");
   for (int i = brojElemenata - 1; i >= 0; i--)
      printf("%.1f\n", *(brojevi + i));

   free(brojevi); // Oslobađanje memorije
   return 0;
}
```

Komentar rješenja 2

- Alociranje memorije za svaki element polja može biti **sporo** zbog (višestrukog) premještanja polja na novu lokaciju u slučaju da nema dovoljno kontinuiranog prostora na trenutnoj lokaciji.
- Moguće rješenje je alociranje memorije u blokovima – npr. za BLOCK_SIZE elemenata odjednom
- Dodati

```
#define VELICINA_BLOKA 100
... //u main
int kapacitet = 0;
```

- Zamijeniti

```
noviBrojevi = realloc(brojevi, (brojElemenata+1) * sizeof(float)); // Dodatna memorija
```

- s

```
if (brojElemenata == 0 || brojElemenata == kapacitet) {
   kapacitet += VELICINA_BLOKA;
   noviBrojevi = realloc(brojevi, kapacitet * sizeof(float));
```

- Mana ovog rješenja je alociranje više memorije nego što je potrebno kada konačan broj elemenata nije višekratnik veličine bloka.
- Ovo je prihvatljivo kompromisno rješenje za brže izvršavanje programa, jer se smanjuje broj učestalih operacija alokacija dodatne memorije (realloc) i potencijalnog premještanja polja.

Modificirano rješenje motivacijskog primjera (2)

```
int main(void) {
   float broj, *brojevi = NULL, *noviBrojevi = NULL;
   int brojElemenata = 0;

   do {
      printf("Upisite %d. broj > ", brojElemenata+1);
      scanf("%f", &broj);
      if (brojElemenata == 0 || brojElemenata*sizeof(float) == kapacitet) {
         kapacitet += VELICINA_BLOKA;
         noviBrojevi = realloc(brojevi, kapacitet * sizeof(float));
         if (noviBrojevi == NULL) {
            printf("Greska u alokaciji memorije.\n");
            free(brojevi);     return 1; //zavrsetak s greskom
         }
         brojevi = noviBrojevi;
      }
      *(brojevi + brojElemenata) = broj; // dodavanje unesenog broja u niz
      brojElemenata++;
   } while (broj != 999);
   printf("Brojevi u obrnutom redoslijedu:\n");
   for (int i = brojElemenata - 1; i >= 0; i--)
      printf("%.1f\n", *(brojevi + i));

   free(brojevi);
   return 0;
}
```

Curenje memorije (memory leak)

- Ako na gomili postoji alocirano memorijsko područje na koje ne pokazuje niti jedan pokazivač, smatra se da je to memorijsko područje "iscurilo".
- Ne postoji mogućnost da se to memorijsko područje oslobodi prije završetka programa.
- Za razliku od nekih drugih programskih jezika, C nema ugrađen tzv. *garbage collector* koji automatski oslobađa memoriju objekata koji se više ne koriste.
- U C-u programer ručno upravlja memorijom i kad je alocira i kad je oslobađa.

Varijabla – pokazivač na memoriju alociranu na gomili je poput vrpce povezane s balonom. Ako ispustite vrpcu (izgubite vrijednost pokazivača), balon (memorija) i dalje negdje postoji (alocirana je i nedostupna za ponovno korištenje), ali se više nećete moći igrati s njim. Također, zagađuje okolinu.

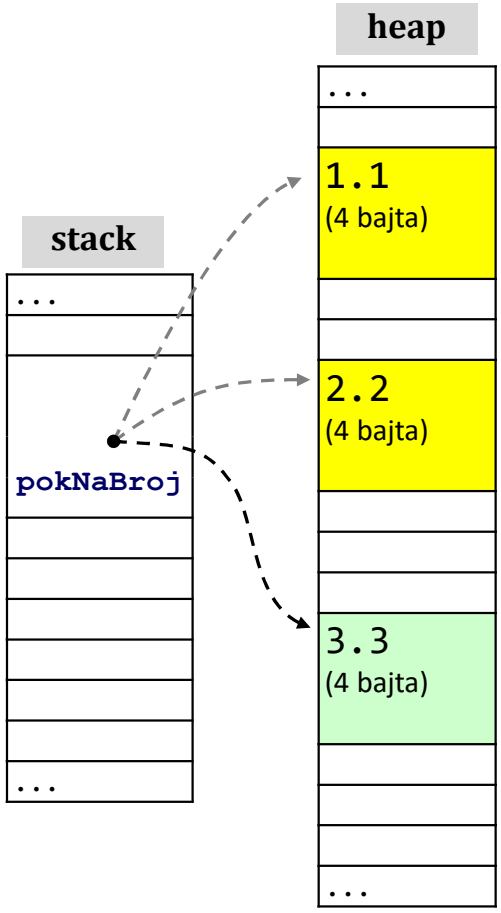
Curenje memorije (*memory leak*)

```
float *pokNaBroj;
for (int i = 1; i <= 3; ++i) {
   pokNaBroj = malloc(sizeof(float));
   *(pokNaBroj) = i * 1.1f;
   printf("%.1f\n", *(pokNaBroj));
}
```

Nakon 3. koraka petlje ne postoje pokazivači na prva dva alocirana područja memorije (a nisu oslobođena). Nitko ih ne može niti koristiti niti osloboditi dok program ne završi. Ta memorija je "iscurila".

Sada je moguće osloboditi samo treće područje memorije. Ta memorija (još) nije "iscurila" jer **postoji mogućnost** njenog oslobađanja.

```
free(pokNaBroj);
```



Spriječiti curenje memorije (*memory leak*)

Napomena:
U praksi se nikada, kao u ovim primjerima, za potrebe pohrane samo jednog realnog broja ne bi koristila dinamički alocirana memorija (treba koristiti automatsku varijablu na stogu).

U primjerima na ovom i prethodnom slajdu to je učinjeno samo radi preglednog prikaza problema curenja memorije.

```
float *pokNaBroj;
for (int i = 1; i <= 3; ++i) {
   pokNaBroj = malloc(sizeof(float));
   *(pokNaBroj) = i * 1.1f;
   printf("%.1f\n", *(pokNaBroj));
   free(pokNaBroj);
} // nema curenja memorije
```

```
float *pokNaBroj;
pokNaBroj = malloc(sizeof(float));
for (int i = 1; i <= 3; ++i) {
   *(pokNaBroj) = i * 1.1f;
   printf("%.1f\n", *(pokNaBroj));
}
free(pokNaBroj);
// nema curenja memorije
```

Curenje memorije (*memory leak*)

- Uočite razliku između donja dva programska odsječka.
- U desnom nije oslobođena dinamički alocirana memorija i dolazi do curenja memorije.

```
float broj, *brojevi = NULL, *noviBrojevi = NULL;
...
noviBrojevi = realloc(brojevi,
                      brojElemenata * sizeof(float));
if (noviBrojevi == NULL) {
   printf("Greska u alokaciji memorije.\n");
   free(brojevi);
   return EXIT_FAILURE;
}
brojevi = noviBrojevi;
```

```
float broj, *brojevi = NULL, *noviBrojevi = NULL;
...
brojevi = realloc(brojevi,
          brojElemenata * sizeof(float));
if (brojevi == NULL) {
   printf("Greska u alokaciji memorije.\n");
   free(brojevi);
   return EXIT_FAILURE;
}
```

- Ako na gomili nema dovoljno slobodne memorije i realloc vrati NULL, inicijalno alocirane memorijske lokacije neće biti oslobođene i za oslobađanje memorije se mora pobrinuti programer.
- Ako na gomili ima dovoljno slobodne memorije, ali ne iza već alocirane memorije, sadržaj do tada alociranih memorijskih lokacija se kopira na novo alocirano područje, a staro područje se oslobađa.

Dinamička alokacija na stogu i na gomili

- Dinamička alokacija memorije na stogu je moguća za VLA polja
- Memorija za VLA polje na stogu automatski se oslobađa završetkom bloka u kojem je polje definirano.
- VLA, za razliku od dinamički alocirane memorije na gomili, ne može mijenjati veličinu uz očuvanje postojećih podataka.
- Gruba procjena maksimalne veličine polja na stogu je 1KB do 2MB
  - Na stogu velikom 8 MB polje od 2MB će zauzeti sljedeća varijabla

```
int polje[524288];
```

- VLA je prikladno koristiti:
  - za polja čija veličina postaje poznata tijekom izvršavanja programa, ali su "razumne" veličine
  - za mala polja kad su performance jako važne, a procjena je da će na stogu biti dovoljno memorije
- **VLA je postao neobavezni dio standarda od C11 pa njegovo korištenje može uzrokovati probleme s portabilnošću**

Dinamička alokacija na stogu i na gomili

- Veličina dinamički alocirane memorije na gomili može se mijenjati uz očuvanje postojećih podataka.
- Memoriju alociranu na gomili potrebno je ručno osloboditi pomoću funkcije *free.*
- Dinamička alokacija memorije na gomili nužna je za stvaranje dinamičkih struktura podataka poput lista i stabala (ASP, sljedeće godine)
- Na gomili je moguće alocirati memoriju reda veličine nekoliko GB

- Dinamičku alokaciju memorije na gomili treba koristiti
  - kada je potrebno mijenjati veličinu alocirane memorije tijekom izvršavanja programa,
  - kada veličina potrebne memorije nadilazi veličinu stoga,

Primjer

- Napišite program koji učitava vrijednosti temperature (tipa float) izmjerenih na različitim mjernim lokacijama u jednoj regiji. Učitavanje prestaje kada korisnik unese temperaturu koja se smatra anomalijom (manja od −100 °C ili veća od 100 °C).
- Program treba ispisati sve učitane temperature i medijan temperature. Medijan odredite pomoću funkcije:

```
float odrediMedijan (float *temp, int broj);
```

- Medijan je vrijednost koja dijeli **sortirani niz podataka** na dva jednako velika dijela. Pola podataka je manje ili jednako medijanu, a pola je veće ili jednako medijanu
  - Ako niz ima neparan broj elemenata → medijan je vrijednost koja se nalazi točno u sredini.
  - Ako niz ima paran broj elemenata → medijan je aritmetička sredina dviju srednjih vrijednosti.
- Primjer:

  Niz: 2, 5, 7 → medijan = 5

  Niz: 1, 4, 6, 9 → medijan = (4 + 6) / 2 = 5

Primjeri izvršavanja programa:

```
Upisite temperature >↵
27.5 30.2 29.0 32.1 28.7 101
Ucitane temperature:↵
27.5 30.2 29.0 32.1 28.7
Medijan: 29.0
```

```
Upisite temperature >↵
27.5 30.2 29.0 32.1 28.7 33.5 -101
Ucitane temperature:↵
27.5 30.2 29.0 32.1 28.7 33.5
Medijan: 29.6
```

Rješenje (1. dio)

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

float pronadjiMedijan(float *temperature, int brojElemenata);

int main(void) {
   float *temperature = NULL, *tmp = NULL, vrijednost;
   int brojElemenata = 0;
   printf("Unesite temperature:\n");
   scanf("%f", &vrijednost);

   while (vrijednost >= -100.0 && vrijednost <= 100.0) {
      tmp = realloc(temperature, (brojElemenata +1) * sizeof(float));
      if (tmp == NULL) {
         printf("Greska: ne mogu realocirati memoriju.\n");
         free(temperature);
         return 1;
      }
      temperature = tmp;
      *(temperature + brojElemenata++) = vrijednost;
      scanf("%f", &vrijednost);
   }
```

Rješenje (2. dio)

```
   printf("\nUnesene temperature:\n");
   for (int i = 0; i < brojElemenata; i++)
      printf("%.2f ", *(temperature + i));

   float medijan = pronadjiMedijan(temperature, brojElemenata);
   printf("\nMedijan temperature: %.1f\n", medijan);

   free(temperature);
   return 0;
}
```

Rješenje (4. dio)

```
float pronadjiMedijan(float *temperature, int brojElemenata) {
   int i, j, ind_min;
   float pomocna;

   for (i = 0; i < brojElemenata - 1; i++) {
      ind_min = i + 1;
      for (j = i + 2; j < brojElemenata; j++)
         if (*(temperature + j) < *(temperature + ind_min)) ind_min = j;

      if (*(temperature + ind_min) < *(temperature + i)) {
         pomocna = *(temperature + i);
         *(temperature + i) = *(temperature + ind_min);
         *(temperature + ind_min) = pomocna;
      }
    }
    if (brojElemenata % 2 == 1) {
       return *(temperature + brojElemenata / 2);
    } else {
       return (*(temperature + brojElemenata / 2 - 1) + *(temperature + brojElemenata / 2)) / 2.0f;
    }
}
```

Primjer

- Napišite funkciju koja će učitavati redak po redak teksta sve dok se kao jedini znak ne učita znak za skok u novi red. Redak teksta uključujući oznaku novog retka, neće biti dulji od 80 znakova. Broj redaka teksta nije unaprijed poznat. Funkcija treba vratiti pokazivač na početak memorije u koju je tekst pohranjen.

```
char *ucitajTekst (void);
```

- Napišite funkciju koja održava "katalog" različitih znakova koji se pojavljuju u učitanom tekstu. Funkcija temeljem ulaznog teksta (*ulazniTekst*) stvara "katalog" (*znakoviUTekstu)* znakova koji se u tekstu pojavljuju, redoslijedom kojim ih pronađe u ulaznom nizu. Znakovi se u "katalogu" ne smiju ponavljati. Funkcija treba vratiti pokazivač na početak memorije u koju je pohranjen "katalog" znakova.

```
char *katalogZnakova (char *ulazniText, char *znakoviUTekstu);
```

Primjer izvršavanja programa:

```
Upisite tekst>
To be, or not to be: that is the question.
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles
↵
Ucitani tekst:
To be, or not to be: that is the question.
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles
Znakovi u tekstu:
To be,rnt:haisqu.
W'lmdfgwOk
```

Rješenje (1. dio)

```
#include <stdio.h>
#include <stdlib.h>
#define MAXNIZ 80
char *ucitajTekst (void);
char *katalogZnakova (char *ulazniText, char *znakoviUTekstu);

int main(void) {
   char *znakoviUTekstu = NULL;
   printf("Upisite tekst>\n");
   char *ulazniTekst = ucitajTekst();
   printf("Ucitani tekst:\n");
   printf("%s", ulazniTekst);
   znakoviUTekstu = katalogZnakova (ulazniTekst, znakoviUTekstu);
   free(ulazniTekst);
   printf("\nZnakovi u tekstu:\n%s", znakoviUTekstu);
   free(znakoviUTekstu);
   return 0;
}
```

Rješenje (2. dio)

```
char *ucitajTekst(void) {
   char recenica[MAXNIZ + 1];
   int duljinaRecenice, duljinaTeksta = 0, i;
   char *tekst = NULL, *pomTekst = NULL;

   fgets(recenica, MAXNIZ + 1, stdin);
   while (*(recenica + 0) != '\n') {
      duljinaRecenice = 0;
      while (*(recenica + duljinaRecenice) != 0) duljinaRecenice++;
      pomTekst = realloc(tekst, duljinaTeksta + duljinaRecenice);
      //provjera ima li memorije... return NULL;
      tekst = pomTekst;
      for (i = 0; i < duljinaRecenice; i++)  *(tekst + duljinaTeksta + i) = *(recenica + i);
      duljinaTeksta += duljinaRecenice;
      fgets(recenica, MAXNIZ + 1, stdin);
   }
   pomTekst = realloc(tekst, duljinaTeksta + sizeof(char));
   //provjera ima li memorije... return NULL;
   tekst = pomTekst;
   *(tekst + duljinaTeksta) = 0;
   return tekst;
}
```

Rješenje (3. dio)

```
char *katalogZnakova(char *ulazniTekst, char *znakoviUTekstu) {
   int duljinaKataloga = 0;
   _Bool znakPostoji;
   char *pomZnakoviUTekstu;

   for (int i = 0; *(ulazniTekst+i) != 0 ; i++) {
      char c = *(ulazniTekst+i);
      znakPostoji = 0;
      for (int j = 0; j < duljinaKataloga; j++) {
         if (*(znakoviUTekstu+j) == c) {
            znakPostoji = 1;
            break;
         }
      }
```

Rješenje (4. dio)

```
      if (!znakPostoji) {
         pomZnakoviUTekstu = realloc(znakoviUTekstu, duljinaKataloga + sizeof(char));
         znakoviUTekstu = pomZnakoviUTekstu;
         *(znakoviUTekstu+duljinaKataloga) = c;
         duljinaKataloga++;
      }
   }
   pomZnakoviUTekstu = realloc(znakoviUTekstu, duljinaKataloga + sizeof(char));
   *(znakoviUTekstu+duljinaKataloga) = 0;

   return znakoviUTekstu;
}
```

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **14. Prije 20. predavanja**
  - **19. vježbe uz predavanja – dinamička alokacija memorije**