Nizovi struktura

- Primer: Napisati program koji broji pojavljivanje pojedinih marki automobila u tekstu sa ulaza
- Bez struktura

```
char *marke[MAXAUTO];
int broj[MAXAUTO];
```

- Sa strukturama

```
struct automobil {
   char *marka;
   int broj;
} lista[MAXAUTO];
```

ili

```
struct automobil {
   char *marka;
   int broj;
};
struct automobil lista[MAXAUTO];
```

Nizovi struktura

- Inicijalizacija

```
struct automobil {
   char *marka;
   int broj;
} lista[] ={
           {"Alfa Romeo", 0},
           {"Audi", 0},
           {"BMW", 0},
           {"Chevrolet", 0},
           {"Fiat", 0},
           {"Ford", 0},
           {"Honda", 0},
                  /* ... */
           {"Renault", 0},
           {"Suzuki", 0},
           {"Toyota", 0},
           {"Volkswagen", 0}
   };
```

Nizovi struktura

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

#define MAXREC 100
#define BRAUTO (sizeof lista / sizeof(struct automobil))
                      ili  #define BRAUTO (sizeof lista / sizeof lista[0])

int getword(char *, int);
int binsearch(char *, struct automobil *, int);

/* brojanje marki automobila */
main()
{
     int n;
     char rec[MAXREC];
     while (getword(rec, MAXREC) != EOF)
         if (isalpha(rec[0]))
               if ((n = binsearch(rec, lista, BRAUTO)) >= 0)
                     lista[n].broj++;
     for (n = 0; n < BRAUTO; n++)
         if (lista[n].broj > 0)
               printf("%4d %s\n", lista[n].broj, lista[n].marka);
     return 0;
}
```

sizeof *objekat*
sizeof (*ime tipa*)

Nizovi struktura

```
/* getword: get next word or character from input */
int getch(void);
void ungetch(int);

int getword(char *word, int lim)
{
        int c;
        char *w = word;

        while (isspace(c = getch()));
        if (c != EOF)
                *w++ = c;
        if (!isalpha(c)) {
                *w = '\0';
                return c;
        }
        for ( ; --lim > 0; w++)
                if (!isalnum(*w = getch())) {
                        ungetch(*w);
                        break;
                }
        *w = '\0';

        return word[0];
}
```

Binarno pretraživanje

```
/* binsearch: trazi marku u nizu tab[0]...tab[n-1] */
int binsearch(char *marka, struct automobil tab[], int n)
{
        int cond;
        int low, high, mid;

        low = 0;
        high = n - 1;

        while (low <= high)
        {
                mid = (low+high) / 2;

                if ((cond = strcmp(marka, tab[mid].marka)) < 0)
                        high = mid - 1;
                else if (cond > 0)
                        low = mid + 1;
                else
                        return mid;
        }

        return -1;
}
```

Pokazivači na strukture

```
/* binsearch: trazi marku u nizu tab[0]...tab[n-1] */
struct automobil *binsearch(char *marka, struct automobil *tab, int n)
{
     int cond;
     struct automobil *low = &tab[0];
     struct automobil *high = &tab[n-1];
     struct automobil *mid;

     while (low < high)
     {
         mid = low + (high-low) / 2;   /* zasto ne moze mid = (low+high) / 2 */

         if ((cond = strcmp(marka, mid->marka)) < 0)
                   high = mid;
         else if (cond > 0)
                   low = mid + 1;
         else
                   return mid;
     }
     return NULL;
}
```

```
struct automobil *p;
...
   if ((p = binsearch(rec, lista, BRAUTO)) != NULL)
        p->broj++;
...
```

Pokazivači na strukture

- Veličina neke strukture ne mora biti zbir veličina njenih članova
- U memoriji rezervisanoj za strukturu mogu postojati neimenovane „rupe“
- Veličina strukture

```
struct{
    char c;
    int i;
};
```

  ne mora biti 5 bajtova, već može zauzimati 8 bajtova
- Operator `sizeof` uvek vraća pravilnu vrednost

TYPEDEF

- C obezbeđuje deklaraciju za definisanje novih tipova:

```
typedef int Length;
Length len, maxlen;
Length *lengths[];

typedef char *String;
String p;

p=(String)malloc(10);
Int strcmp(String,String);
```

- Deklaracija `typedef` ne stvara novi tip, već samo dodaje novo ime za neki postojeći tip.

TYPEDEF

- Složenije deklaracije:

```
typedef struct node *Treeptr;

typedef struct tnode {
    char *word;
    int count;
    struct tnode *left;
    struct tnode *right;
} Treenode;

Treeptr talloc(void){
    return (Treeptr)malloc(sizeof(Treenode));
}
```

TYPEDEF

- Složenije deklaracije:

```
typedef int (*PFI)(char *, char *);
```

PFI je pokazivač na funkciju (od dva ragumenta tipa `char *`) koja vraća tip `int`

```
PFI strcmp, numcmp
```

Unije

- Unija je promenljiva koja može čuvati u različito vreme objekte različitih tipova i veličina.

```
union primer {
    int broj;
    char slovo;
    float broj_r;
} u;
```

- Promenljiva u biće dovoljno velika kako bi mogla da sadrži najveći od ova tri tipa.

- Programer je obavezan da vodi računa o tome koji tip vrednosti se trenutno čuva u uniji.

Unije

- Članovima unije se pristupa isto kao članovima strukture:

  ime-unije.član ili *pokazivač-na-uniju->član*

- Unije se mogu koristiti unutar struktura i nizova i obratno.

```
struct{
    char *name;
    int flags;
    union {
        int ival;
        char *sval;
    } u;
} symtab[NSYM];

symtab[i].u.ival
*symtab[i].u.sval    ili   symtab[i].u.sval[0]
```

Unije

- Za unije su dozvoljene iste operacije kao za strukture (kopiranje vrednosti, dodeljivanje neke vrednosti, uzimanje adrese, pristupanje članu)
- Unija se može inicijalizovati samo vrednošću tipa njenog prvog člana.

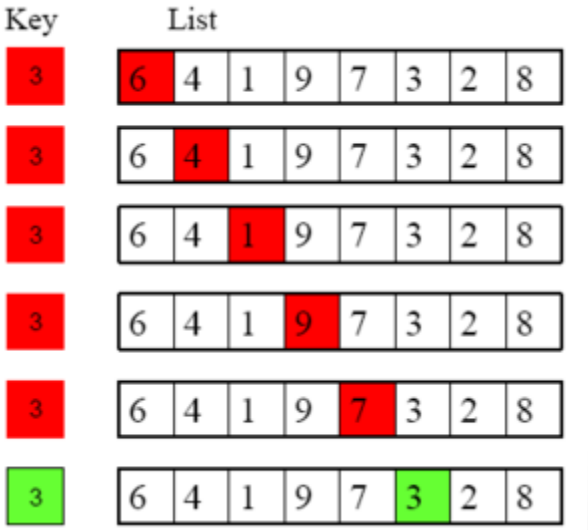
```
struct radnik {
    char prezime[20];
    char ime[20];
    char plata_ili_nadnica;
    union {
        float plata;
        struct {
            int sati_rada;
            float satnica;
        } nadnica;
    } zarada;
} osoba[20];
```

PRETRAŽIVANJE NIZOVA

Linearno pretraživanje

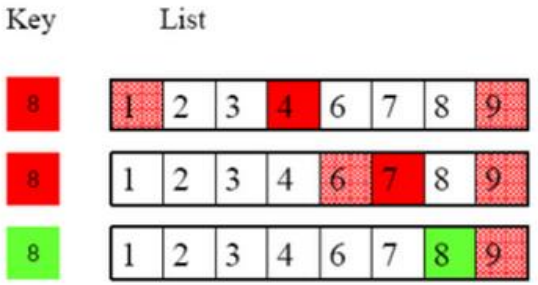
- U nizu elemenata pronaći element sa odgovarajućom vrednošću pretražujući niz od početka, jedan po jedan element

```
int LinearSearch(int list[], int n, int key) {
  for (int i = 0; i < n; i++)
      if (key == list[i])
          return i;
   return -1;
}
```



Binarno pretraživanje

- Efikasan algoritam za pretraživanje sortiranog niza
- Traženi ključ se upoređuje sa središnjim elementom niza ili elementom na kraju prve polovine niza u slučaju niza parne dužine
  - Ukoliko je ključ manji od središnjeg elementa, potrebno je pretražiti samo prvu polovinu niza, po istom principu
  - Ukoliko je ključ veći od središnjeg elementa potrebno je pretražiti samo drugu polovinu niza
  - Ukoliko je ključ jednak središnjem elementu, staje se sa zaključkom da je element pronađen



Binarno pretraživanje

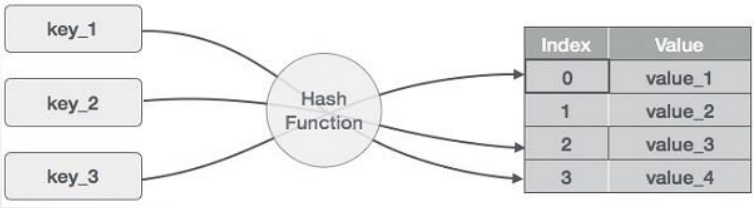
```
int BinSearch(int list[], int n, int key)
{
   int low, high, mid;

   low = 0;
   high = n - 1;

   while (low <= high)
   {
      mid = (low+high)/2;

      if (key < list[mid])
         high = mid - 1;
      else if (key > list[mid])
         low = mid + 1;
      else /* found match */
         return mid;
   }

   return -1; /* no match */
}
```

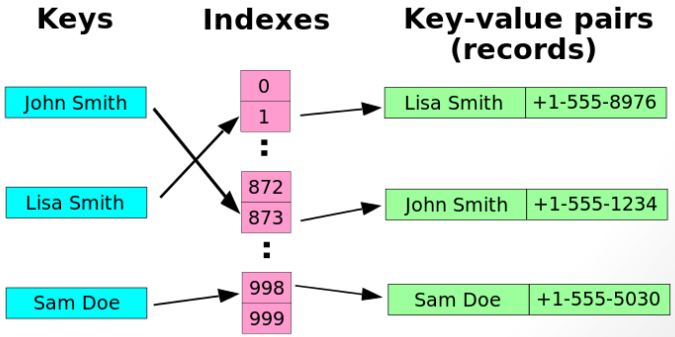
Heš tabela (Hash table)

- Ideja heširanje je nastali nezavisno na različitim mestima.
  - 1953, HP-Luhn napisao je interni memorandum da koristi heširanje sa lančanim listama.
  - U isto vreme Gn Amdahl, EM Boehme, N. Shirley i AL Semjuel implementirali su programa za heširanje.
  - Otvoreno adresiranje sa linearnom pretragom pripisano je Amdahlu i ako je Ershov (Rusija) imao istu ideju.
- Heš tabela je strukture podataka koje koriste heš funkciju za efikasno preslikavanje određenih ključeva u njima pridružene vrednosti.
- U heš tabelama podaci su spakovanu formatu niza gde svaka vrednost ima jedinstveni indeks.
- Pristup podacima je jako brz ako je poznat indeks traženog elemeta.

Heš tabela (Hash table)

- Heš funkcija se koristi za transformisanje ključa u indeks (heš) to jest mesto u nizu elemenata gde treba tražiti odgovarajuću vrednost.
- Idelna heš funkcija preslikava svaki mogući ključ u zaseban indeksi (1-1), što je u praksi najčešće nemoguće.
- Većina implementacija heš tabela podrazumava postojanje kolizija, tj. parova različitih ključeva sa istim heš vrednostima.
- Hash funkcija treba da bude što prostija zbog lakšeg i bržeg izračunavanja
- U mnogim situacijama, heš tabele se pokazuju efikasnijim od stabala pretrage ili drugih tabelarnih struktura, zbog čega se nalaze u širokoj upotrebi





Izbegavanje kolizije

- Lančanje
  - Svi sudari se povezuju u listu koja se „prikači“ na isto polje.
  - Može se rukovati negraničenim brojem sudara, ali implementacija povezanih lista zahteva više prostora i vremena od nizova
- Linerano pretraživanje
  - Kada nastane sudar vrednost se smešta/traži u narednom polju
  - Nova adresa se brzo računa i pristup je vrlo efikasan
- Dvostruko heširanje
  - Koriste se dve heš funkcije
    $h(k,i)=(h1(k)+i\cdot g(k))\%n, \quad i=0,1,\ldots,n-1$

Izbegavanje kolizije

- Primer. U heš tabelu veličine n = 11 smestiti vrednosti {22, 1, 13, 11, 24, 33, 18, 42, 31, 75} u zadatom poretku.

  Primarna heš funkcija: $h1(x)=x\%11$

  Za potrebe dvostrukog heširanje: $g(x)=1+x\%10$

Pozicija	Lančanje	Lin. pretražinanje	Dvostruko heš.
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Izbegavanje kolizije

- Primer. U heš tabelu veličine n = 11 smestiti vrednosti {22, 1, 13, 11, 24, 33, 18, 42, 31, 75} u zadatom poretku.

  Primarna heš funkcija: $h1(x)=x\%11$

  Za potrebe dvostrukog heširanje: $g(x)=1+x\%10$

Pozicija	Lančanje	Lin. pretražinanje	Dvostruko heš.
0	33 -> 11 -> 22		
1	1		
2	24 -> 13		
3			
4			
5			
6			
7	18		
8			
9	75 -> 31 -> 42		
10			

Izbegavanje kolizije

- Primer. U heš tabelu veličine n = 11 smestiti vrednosti {22, 1, 13, 11, 24, 33, 18, 42, 31, 75} u zadatom poretku.

  Primarna heš funkcija: $h1(x)=x\%11$

  Za potrebe dvostrukog heširanje: $g(x)=1+x\%10$

Pozicija	Lančanje	Lin. pretražinanje	Dvostruko heš.
0	33 -> 11 -> 22	22	
1	1	1	
2	24 -> 13	13	
3		11	
4		24	
5		33	
6		75	
7	18	18	
8			
9	75 -> 31 -> 42	42	
10		31	

Izbegavanje kolizije

- Primer. U heš tabelu veličine n = 11 smestiti vrednosti {22, 1, 13, 11, 24, 33, 18, 42, 31, 75} u zadatom poretku.

  Primarna heš funkcija: $h1(x)=x\%11$

  Za potrebe dvostrukog heširanje: $g(x)=1+x\%10$

Pozicija	Lančanje	Lin. pretražinanje	Dvostruko heš.
0	33 -> 11 -> 22	22	
1	1	1	
2	24 -> 13	13	
3		11	
4		24	
5		33	
6		75	
7	18	18	
8			
9	75 -> 31 -> 42	42	
10		31	

$h(x,i)=(h1(x)+i\cdot g(x))\%11$
22 -> $h(22,0)=(0+0\cdot3)\%11 = 0$
1 -> $h(1,0)=(1+0\cdot2)\%11 = 1$
13 -> $h(13,0)=(2+0\cdot4)\%11 = 2$
11 -> $h(11,0)=(0+0\cdot2)\%11 = 0$
$h(11,1)=(0+1\cdot2)\%11 = 2$
$h(11,2)=(0+2\cdot2)\%11 = 4$
24 -> $h(24,0)=(2+0\cdot5)\%11 =2$
$h(24,1)=(2+1\cdot5)\%11 =7$
⋮

Izbegavanje kolizije

- Primer. U heš tabelu veličine n = 11 smestiti vrednosti {22, 1, 13, 11, 24, 33, 18, 42, 31, 75} u zadatom poretku.

  Primarna heš funkcija: $h1(x)=x\%11$

  Za potrebe dvostrukog heširanje: $g(x)=1+x\%10$

Pozicija	Lančanje	Lin. pretražinanje	Dvostruko heš.
0	33 -> 11 -> 22	22	22
1	1	1	1
2	24 -> 13	13	13
3		11	
4		24	11
5		33	18
6		75	31
7	18	18	24
8			33
9	75 -> 31 -> 42	42	42
10		31	75

$h(x,i)=(h1(x)+i\cdot g(x))\%11$
22 -> $h(22,0)=(0+0\cdot3)\%11 = 0$
1 -> $h(1,0)=(1+0\cdot2)\%11 = 1$
13 -> $h(13,0)=(2+0\cdot4)\%11 = 2$
11 -> $h(11,0)=(0+0\cdot2)\%11 = 0$
$h(11,1)=(0+1\cdot2)\%11 = 2$
$h(11,2)=(0+2\cdot2)\%11 = 4$
24 -> $h(24,0)=(2+0\cdot5)\%11 =2$
$h(24,1)=(2+1\cdot5)\%11 =7$
⋮

Realizacija heš tabele

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

#define SIZE 20

struct DataItem {
   int data;
   int key;
};

struct DataItem* hashArray[SIZE];
struct DataItem* dummyItem;
struct DataItem* item;

int hashCode(int key) {
   return key % SIZE;
}
```

Realizacija heš tabele - insert

```
void insert(int key,int data) {

   struct DataItem *item = (struct DataItem*)malloc(sizeof(struct DataItem));
   item->data = data;
   item->key = key;

   int hashIndex = hashCode(key);

   while(hashArray[hashIndex] != NULL &&
         hashArray[hashIndex]->key != -1) {

      ++hashIndex;
      hashIndex %= SIZE;
   }

   hashArray[hashIndex] = item;
}
```

Realizacija heš tabele - search

```
struct DataItem *search(int key) {

   int hashIndex = hashCode(key);

   while(hashArray[hashIndex] != NULL) {
      if(hashArray[hashIndex]->key == key)
          return hashArray[hashIndex];

      ++hashIndex;
      hashIndex %= SIZE;
   }
   return NULL;
}
```

Element ako postoji u tabeli mora da se nađe pre prve slobodne pozicije

Realizacija heš tabele - delete

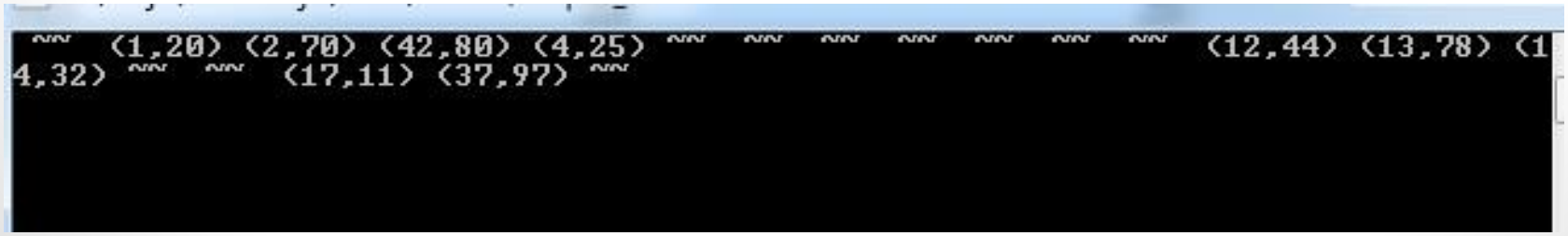
```
struct DataItem* delete(struct DataItem* item) {
   int key = item->key;
   int hashIndex = hashCode(key);

   while(hashArray[hashIndex] != NULL) {

      if(hashArray[hashIndex]->key == key) {
         struct DataItem* temp = hashArray[hashIndex];
         hashArray[hashIndex] = dummyItem;
         return temp;
      }

      ++hashIndex;
      hashIndex %= SIZE;
   }
   return NULL;
}
```

Realizacija heš tabele - display

```
void display() {
   int i = 0;

   for(i = 0; i<SIZE; i++) {

      if(hashArray[i] != NULL)
         printf(" (%d,%d)",hashArray[i]->key,hashArray[i]->data);
      else
         printf(" ~~ ");
   }

   printf("\n");
}
```

Realizacija heš tabele - main

```
int main() {
   dummyItem = (struct DataItem*) malloc(sizeof(struct DataItem));
   dummyItem->data = -1;
   dummyItem->key = -1;

   insert(1, 20);
   insert(2, 70);
   insert(42, 80);
   insert(4, 25);
   insert(12, 44);
   insert(14, 32);
   insert(17, 11);
   insert(13, 78);
   insert(37, 97);

   display();
```



Realizacija heš tabele - main

```
    item = search(37);
    if(item != NULL) {
       printf("Element found: %d\n", item->data);
    }
    else {
       printf("Element not found\n");
    }
    delete(item);

    item = search(37);
    if(item != NULL) {
       printf("Element found: %d\n", item->data);
    }
    else {
       printf("Element not found\n");
    }
}
```

```
~~  (1,20) (2,70) (42,80) (4,25) ~~  ~~  ~~  ~~  ~~  ~~  ~~  (12,44) (13,78) (1
4,32) ~~  ~~  (17,11) (37,97) ~~
Element found: 97
Element not found
```