# Linearno pretraživanje polja

- Napiši funkciju Search koja uzima sljedeće ulazne parametre:
  - polje cijelih brojeva a[]
  - duljinu length polja a[]
  - cijeli broj x

  Funkcija kao rezultat svog rada treba vratiti indeks onog elementa polja a[] u kojemu se nalazi broj x, odnosno -1 ukoliko se broj x ne nalazi u polju a[].

- Pretpostavimo da imamo polje **nesortiranih** elemenata i da želimo znati nalazi li se ili ne nalazi određeni zadani element u tom polju i ako se nalazi, želimo znati njegovu poziciju (indeks) u tom polju. Obzirom na to da polje nije sortirano, ne preostaje nam ništa drugo nego da "obiđemo" polje element po element i uspoređujemo vrijednost svakog elementa s brojem koji tražimo. Pri tom obilasku mogu nam se dogoditi 2 stvari:

  u nekom trenutku možemo naići na element u polju koji ima vrijednost koju tražimo. Tada zapamtimo njegov indeks i prestajemo s radom, ili

  došli smo do kraja polja i nigdje u njemu nismo našli traženi broj. Tada na odgovarajući način signaliziramo da traženi broj nije nađen i prestajemo s radom.

  Mana ovakvog pretraživanja je da za pronalaženje elementa moramo u najgorem slučaju obići čitavo polje, što može biti sporo za jako velika polja.

```
#include <stdio.h>
#include <stdlib.h>
int Search(int a[], int length, int key);

int main () {
      int N, i, trazeni, rez;
      int *polje;
      printf("Unesi duljinu polja:\n");
      scanf("%d",&N);
            if ((polje = (int *)malloc(N*sizeof(int))) == NULL){
              printf("Greska u alociranju memorije !");
             exit (1); }
       for (i = 0; i<N; i++)        {
            printf("Unesi %d. broj:",i);
            scanf("%d", &polje[i]);  }
        printf("\n");
        printf("Unesi vrijednost koju trazis:");
        scanf("%d", &trazeni);
       rez = Search(polje, N, trazeni);
       if (rez != -1)
           printf("Broj %d se nalazi u polju na poziciji %d\n", trazeni,rez);
       else
           printf("Broj %d se ne nalazi u polju\n", trazeni);
       system("PAUSE"); return 0;
       }
```

```
int Search(int a[], int lenght, int key)   {
            int i;
            for (i = 0; i < lenght; i++)
                if (a[i] == key) return i;
                        return -1;
        }
```

# Pretraživanje niza znakova

- Napravite niz duljine do 80 znakova koje unesete, upišete znak koji tražite u nizu i napišite program koji pretražuje niz znakova i javlja poziciju na kojoj se traženi znak nalazi, ukoliko se traženi znak ne nalazi u nizu, obavještava Vas o tome

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define L 80
int traziznak(char *, char *, int);

int main(){
      char *tekst;
      char *ctrazeni;
      int poz=0;
      int nizl;
      if ((tekst = (char *)malloc(L+1)) == NULL)
            {printf("Greska u alociranju memorije !");
          exit (1); }
if ((ctrazeni = (char *)malloc(2)) == NULL)
            {printf("Greska u alociranju memorije !");
          exit (1); }
```

```
    printf("Unesi do %d znaka:\n",L);
    gets(tekst);
    printf("Unesi znak koji trazis:\n");
    gets(ctrazeni);
    nizl=strlen(tekst);
    printf("Uneseni niz ima %d znakova\n",nizl);

    poz = traziznak(tekst,ctrazeni,nizl);
    if (!poz)
            printf("Znak %s se ne nalazi u nizu %s\n",ctrazeni,tekst);
            else
    printf ("Znak %s je na mjestu %d u nizu %s\n",ctrazeni,poz,tekst);
    system("PAUSE");
    return 0;
}

int traziznak(char *niz, char *znak, int nizl){
    int ind;
    for (ind=0; ind <= nizl; ind++)
            if (niz[ind] == *znak) return ind+1;
    return 0;
}
```

# Dvodimenzionalna i višedimenzionalna polja kao argumenti funkcije

- Polje u funkciji treba prihvatiti kao jednodimenzionalno polje ili pokazivač.
- Primjer:

```
int polje[3][4] = { { 1, 2, 3, 4},
                    { 5, 6, 7, 8},
                    { 9, 10, 11, 12} };
```

u memoriji računala spremljeno je kao


tj. jednako kao jednodimenzionalno polje od 12 elemenata.

- Razlikuju se fizičke dimenzije polja od logičkih, koje mogu biti i manje. Za pozicioniranje je potrebno znati fizički broj stupaca (`MAX_broj_stupaca`).
- Za dohvat elementa iz *i*-tog retka treba preskočiti *i*`-1` punih redaka.
- Kako prvi redak ima indeks 0, drugi 1, treći 2 itd., za dohvat elemenata retka s indeksom *i* treba preskočiti *i*`*MAX_broj_stupaca` članova polja.
- Općenito:

```
dvodim_polje[i][j] ≡ jednodim_polje[i * MAX_broj_stupaca + j]
```

# Zbrajanje i množenje matrica

- Napišite program koji će rezervirati u memoriji mjesto za tri dvodimenzionalna polja (matrice) A, B i C dimenzija maxstu. Polja A i B popuniti slučajno generiranim cijelim brojevima u rasponu od 0 do maxcl. Napisati funkciju koja zbroji matrice A i B i rezultat pospremi u polje C. Napisati funkciju koja umnožak matrica A*B pospremi u polje C. Ispisati sva polja.

- Suma: $c_{ij} = a_{ij} + b_{ij}$

- Produkt: $$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#define maxstu 5
#define maxcl 9

void prikazi(int *M){
int i,j;

for (i=0;i<maxstu;i++)  {
   for (j=0;j<maxstu;j++)
       printf("%d ",*(M+i*maxstu+j));
   printf("\n");
  }
printf("\nPritisnite Enter za nastavak!\n");
scanf("%*c");
}

void zbroji(int *A,int *B,int *C){
int i,j;

 for (i=0;i<maxstu;i++)
    for (j=0;j<maxstu;j++)
      *(C+i*maxstu+j)=*(A+i*maxstu+j)+*(B+i*maxstu+j);
}
```

```
void mnozi(int *A,int *B,int *C){
int i,k,j;

 for (i=0;i<maxstu;i++)
    for (j=0;j<maxstu;j++)
       for (k=0;k<maxstu;k++)
     *(C+i*maxstu+j)+=*(A+i*maxstu+k) * *(B+k*maxstu+j);
}

int main(){
int i,j;
int A[maxstu][maxstu];
int B[maxstu][maxstu];
int C[maxstu][maxstu]={0};

srand(time(NULL));
for (i=0;i<maxstu;i++)
   for (j=0;j<maxstu;j++)   {
      srand(time(NULL) * (float) rand() / (RAND_MAX+1));
      A[i][j]=maxcl * ((float) rand() / (RAND_MAX+1));
      B[i][j]=maxcl * ((float) rand() / (RAND_MAX+1));
   }
```

```
printf("\n------------------A------------------\n\n");
prikazi(&A[0][0]);
printf("\n------------------B------------------\n\n");
prikazi(&B[0][0]);
printf("\n------------------C------------------\n\n");
prikazi(&C[0][0]);

zbroji(&A[0][0],&B[0][0],&C[0][0]);
printf("\n----------------C=A+B------------------\n\n");
prikazi(&C[0][0]);

for (i=0;i<maxstu;i++)
  for (j=0;j<maxstu;j++)
      C[i][j]=0;

mnozi(&A[0][0],&B[0][0],&C[0][0]);
printf("\n----------------C=A*B------------------\n\n");
prikazi(&C[0][0]);

return 0;
}
```

# Primjeri rekurzija

- Primjeri jednostavnih rekurzivnih funkcija za koje treba odrediti što će ispisati i koju će vrijednost sadržavati varijabla nula nakon povratka u glavnu proceduru. Također odgovoriti na pitanja:
  a) Kakav tip podatka sadrži *broj?
  b) Kakav tip podatka sadrži broj?
  c) Kakav tip podatka sadrži &nula?

```
#include <stdio.h>
#include <stdlib.h>
void pisi1 (int broj, int n) {
  broj++;
  if (broj > n) return;
  pisi1 (broj, n);
  printf( "%d ", broj);
}
void pisi2 (int broj, int n) {
  broj++;
  if (broj > n) return;
  printf( "%d ", broj);
  pisi2 (broj, n);
}
void pisi3 (int *broj, int n) {
  (*broj)++;
  if (*broj > n) return;
  pisi3 (broj, n);
  printf( "%d ", *broj);
}
void pisi4 (int *broj, int n) {
  (*broj)++;
  if (*broj > n) return;
  printf( "%d ", *broj);
  pisi4 (broj, n);
}
```

```
void pisi5 (int *broj, int n) {
  (*broj)--;
  if (*broj < 0) return;
  pisi5 (broj, n);
  printf( "%d ", *broj);
}

void main (void) {
  int nula;
      nula = 0; pisi1 (nula, 10);
      printf(" Nakon pisi1 nula = %d\n", nula);
      nula = 0; pisi2 (nula, 10);
      printf(" Nakon pisi2 nula = %d\n", nula);
       nula = 0; pisi3 (&nula, 10);
      printf(" Nakon pisi3 nula = %d\n", nula);
      nula = 0; pisi4 (&nula, 10);
      printf(" Nakon pisi4 nula = %d\n", nula);
      nula=10; pisi5 (&nula, 10);
      printf(" Nakon pisi5 nula = %d\n", nula);
     system("PAUSE");
      exit (0);
}
```

# Još primjera rekurzija

- Nekoliko jednostavnih primjera rekurzija s predavanja:
  - Rekurzivno traženje indeksa člana u polju
  - Rekurzivno traženje indeksa člana u polju s ograničivaćem
  - Rekurzivno traženje najvećeg člana polja
  - Rekurzivno traženje najvećeg člana polja – strukturirano
  - Primjer neispravne rekurzije

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXA 15

// ispis znaka c u zadanoj duljini n
void nznak (int c, int n) {
  while (--n >= 0) putchar(c);
}

// ispis polja
void ispisi(int A[], int n) {
      int i;
      printf("\n");
      for (i = 0; i < n; i++) printf(" A[%d]",i);
      printf("\n");
      for (i = 0; i < n; i++) printf("%5d", A[i]);
      printf("\n");
}
```

```
// Rekurzivno trazenje indeksa clana u polju
int trazi (int A[], int x, int n, int i) {     // A-polje, x-trazeni, i-indeks od kojeg se trazi
      int ret;
      nznak(' ', i*5); printf("^^^^^\n");
      if (i >= n)
              ret = -1;
      else if (A[i] == x)
              ret = i;
      else
             ret = trazi (A, x, n, i+1);
      nznak(' ', i*5); printf("%5d\n", ret);
  return ret;
}

// Rekurzivno trazenje indeksa clana u polju s ogranicivacem
int trazi1 (int A[], int x, int i){
      int ret;
      nznak(' ', i*5); printf("^^^^^\n");
      if(A[i] == x)
              ret = i;
      else
              ret= trazi1 (A, x, i+1);
  nznak(' ', i*5); printf("%5d\n", ret);
  return ret;
}
```

```
// Rekurzivno trazenje najveceg clana polja
int maxclan (int A[], int i, int n) {
  int imax;
  if (i >= n-1)
        return n-1;
  imax = maxclan (A, i + 1, n);
  if (A[i] > A[imax])
        return i;
  return imax;
}

// Rekurzivno trazenje najveceg clana polja - strukturirano
int maxclan1 (int A[], int i, int n) {
  int imax, ret;
     printf ("max(%d) -> ", i);
  if (i >= n-1) {
             printf ("\n");
             ret = n-1;
     } else {
          imax = maxclan1 (A, i + 1, n);
          if (A[i] > A[imax])
              ret = i;
          else
             ret = imax;  }
     printf ("<- max(%d)=%d ", i, ret);
  return ret; }
```

```
// macro naredba za vecu od dvije vrijednosti
#define maxof(a,b) ((a) > (b) ? (a) : (b))

// Funkcija s macro naredbom koja vraca vrijednost najveceg clana
int maxclan2 (int A[], int i, int n) {
      int m;
  if (i >= n-1) return A[i];
      m = maxclan2 (A, i + 1, n);
  return maxof(A[i], m);
}

// Primjer neispravne rekurzije
int los (int n, int *dubina) {
      int r;
      (*dubina)++;
      printf ("n = %d, dubina rekurzije = %d\n", n, *dubina);
      if (n == 0)
            r = 0;
      else
            r = los (n / 3 + 1, dubina) + n - 1;
  return r;
}
```

```
void main () {
  int A[MAXA], x, i, n, dubina;
  FILE *fi;
  fi = fopen ("UlazZaRekurzije.txt", "r");
  if (!fi) exit (1);
  n = 0;
  while (n < MAXA - 1 && fscanf (fi, "%d", &A[n]) != EOF)    n++;
  fclose (fi);
  ispisi (A, n);
     printf ("Upisite vrijednost za x =");  scanf ("%d", &x);
     printf ("\nRekurzivno trazenje indeksa clana\n");
     ispisi (A, n);
     if ((i = trazi (A, x, n, 0)) < 0) {
              printf ("Vrijednost %d ne postoji u polju\n", x);
     } else {
              printf ("A [%d] = %d\n", i, A [i]);          }
  printf ("\nRekurzivno trazenje ... s ogranicivacem\n");
     A [n] = x;  // postavljanje ogranicivaca
     ispisi (A, n+1);
     if ((i = trazi1 (A, x, 0)) == n) {
              printf ("Vrijednost %d ne postoji u polju", x);
     } else {
              printf ("A [%d] = %d\n", i, A [i]);          }
```

```
printf ("\nRekurzivno trazenje najveceg...\n");
      ispisi(A, n);
      if ((i = maxclan (A, 0, n)) != maxclan1 (A, 0, n)) {
            printf ("Pogreska: Strukturirana i nestrukturirana funkcija daju razlicite rezultate!\n");
            exit (0);
      }
  printf ("\nNajveci clan A [%d] = %d\n",i, A [i]);
printf ("Funkcija s macro naredbom je nasla najveci clan %d\n", maxclan2 (A, 0, n));

      printf ("\nPozivam neispravnu rekurziju\n");
            dubina = 0;
            printf ("Upisite vrijednost za n =");
            scanf ("%d", &n);
            i = los (n, &dubina);
            printf ("\ni = %d", i);
      exit(0);
}
```

# Algoritam binarnog pretraživanja

- Za pretraživanje uzlazno sortiranog polja koristimo algoritam koji je bitno efikasniji (brži) od linearnog pretraživanja. Njegov rad opisat ćemo na primjeru:
- Pretpostavimo da u *sortiranom* polju

  2, 5, 7, 8, 10, 17, 19, 23, 25, 27, 33, 35, 36, 40, 42, 45, 53

  tražimo element 17.
- Prvo pogledamo element koji se nalazi u sredini polja (to je element 25) i usporedimo ga s brojem kojeg tražimo (broj 17). Kako je 17 manji od 25, zaključujemo da se 17 nikako ne može nalaziti desno od 25 (jer je polje uzlazno sortirano!) pa elemente 27, 33, 35, 36, 40, 42, 45, 53 ne trebamo niti gledati – svi su oni veći od 17. Stoga u drugom koraku gledamo samo podniz koji se nalazi lijevo od 25:

  2, 5, 7, 8, 10, 17, 19, 23
- Sada pogledamo srednji element ovog podniza. Srednji element u ovom slučaju je 8. Kako je 17 veći od 8, zaključujemo da se 17 nikako ne može nalaziti lijevo od srednjeg elementa 8, pa elemente lijevo od 8 (2, 5 i 7) ne moramo ni gledati, nego u trećem koraku gledamo podniz desno od 8:

  10, 17, 19, 23
- Pogledamo srednji element ovog podniza. No, srednji element je upravo jednak traženom elementu 17, pa algoritam završava s radom jer je pronašao ono što je tražio.

```
#include <stdio.h>
#include <stdlib.h>
int BinarySearch(int a[], int length, int key);

void main(){
    int n, i, k;
    int *niz;
    int rezultat;
    printf("Unesi duljinu polja:\n");
    scanf("%d",&n);
    if ((niz = (int *)malloc(n*sizeof(int))) == NULL) {
        printf("Greska u alociranju memorije !");
        exit (1);
            }

    for(i = 0; i < n; i++)
            niz[i] = 2*i + 1;
    printf("Elementi niza su:\n");
    for(i = 0; i < n; i++)
            printf("%d ", niz[i]);

    printf("\nUnesi element pretrage: ");
    scanf("%d", &k);
    rezultat = BinarySearch(niz, n, k);
```

```
    if (rezultat == -1)
            printf("Uneseni element se ne nalazi u nizu!\n");
    else
            printf("Uneseni element se nalazi na poziciji %d", rezultat+1);
    system("PAUSE");
  }

int BinarySearch(int a[], int length, int key) {
    int left = 0;
    int right = length - 1;
    int middle;
    do    {
            middle = (left + right) / 2;
            if (a[middle] == key)
                    return middle;
            else if (a[middle] < key)
                    left = middle + 1;
            else
                    right = middle - 1;
    }  while (left <= right);
    return -1;
}
```

- Efikasnost algoritma pretraživanja:
- Da bi pronašli traženi element u sortiranom nizu duljine $n=2^m$ koristeći algoritam binarnog pretraživanja potrebno je u najgorem slučaju m+1 korak
- $2^m \rightarrow 2^{m-1} \rightarrow \ldots \rightarrow 2^1 \rightarrow 2^0$ koraka
- Za niz duljine $2^{m-1} < n < 2^m$ potrebno je također najviše m+1 koraka

- Za pronalaženje elementa u nizu duljine n upotrebom linearnog pretraživanja potrebno je u najgorem slučaju n koraka

- Npr. n= 128, algoritam binarnog pretraživanja treba maksimalno 8 koraka, a algoritam linearnog pretraživanja 128 koraka

# Pronalaženje najvećeg elementa u nesortiranom polju

- Napisati program koji će pronaći najveći broj u polju slučajno generiranih cijelih brojeva. Generirane brojeve ispisati u datoteku.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define N 100

int main() {
    short int a[N], max, i;
    FILE *fpout;

    if ((fpout=fopen("proba.txt","w"))==NULL) {
                    printf("Greska u otvaranju datoteke!\n");
                      exit(1);
          }
```

```
printf ("Generirat cu %d brojeva izmedju 0 i %d\n",N,SHRT_MAX/N);

srand(time(NULL));
for(i = 0; i < N; i++) {
        srand(time(NULL)/rand());
        a[i]= SHRT_MAX/N *((float) rand()/(RAND_MAX + 1));
        fprintf(fpout,"%d\n",a[i]);
}

max = a[0];
for(i = 1; i < N; i++)
        if (a[i] > max)
                max = a[i];

printf("Najveci uneseni broj je %d\n", max);
system("PAUSE");
return 0;
}
```