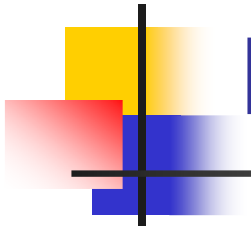
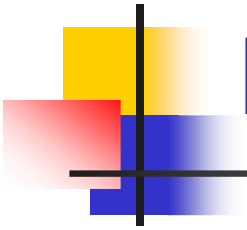
Показивачи: аритметика

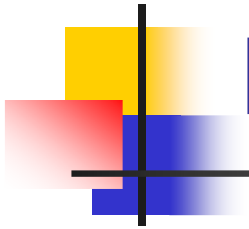


- Pokazivači: adrese objekata
- Grupa susednih bajtova (obično dva ili četiri) koja može da sadrži adresu.
- Npr. deklaracija
  - char *vrednost;
  - struct slog *sp;
  - vrednost, sp – adrese;
  - *vrednost – char, *sp – struktura
  - *(sp).kljuc (ili, ekvivalentno, *sp->kljuc)* - ceo broj
- Operator * : dereferenciranja (uzimanja vrednosti)
- Operator &: referenciranja (uzimanja adrese)
  - sp = &primerak sloga;



Показивачи: аритметика

- operator *referenciranja* *&* daje adresu objekta - operanda na koji se primenjuje.
- Ako je c promenljiva tipa char a p - pokazivač koji pokazuje na karakter, iskaz
- p = &c;
- dodeljuje promenljivoj p adresu promenljive c, i kaže se da p "pokazuje na" c
- Operator referenciranja & moze da se primeni samo na objekte sa adresom, tj. na promenljive i elemente nizova, i ne može da se primeni, na primer, na izraze ili konstante.



Показивачи: аритметика

- Inverzni operator, *dereferenciranja* ili *indirekcije* , primenjuje se na pokazivač i daje sam objekat na koji pokazivač pokazuje.
- Deklaracija pokazivačke promenljive označena je tipom na koji pokazivač pokazuje
- Pokazivač ip koji može da pokazuje na promenljivu celobrojnog tipa deklariše se sa:
- int *ip;
- a pokazivač p koji pokazuje na promenljivu znakovnog tipa deklariše se sa
- char *p;

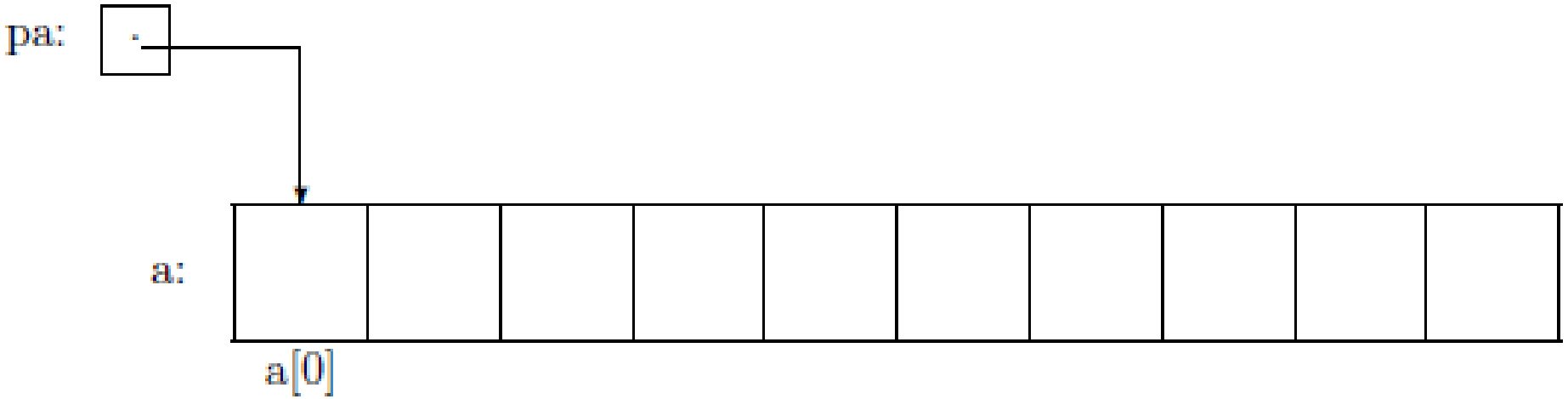
Показивачи: аритметика

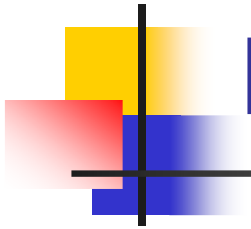
- Operacije: poređenje, oduzimanje; dodavanje, oduzimanje celog broja
- aritmetika*:*
  - *vrednost* + 1 – adresa sledećeg karaktera
  - *sp* + 1 – adresa sledećeg sloga
- Primer:
- int x=1, y=2, z[10];
- int *ip; /* *ip je pokazivac na int */*
- ip = &x; /* *ip sada pokazuje na x */*
- y = *ip; /* *y sada ima vrednost 1 */*
- *ip = 0; /* *x sada ima vrednost 0 */*
- ip = &z[0]; /* *ip sada pokazuje na z[0] */*
- *ip += 1; /* *kao i ++*ip; i (*ip)++; povećava vrednost promenljive na koju*
- pokazuje ip */

Показивачи и низови

- Pokazivači i nizovi – sinonimi u C-u
- Npr. *A*[0] ili **A*
- char *vrednost
- vrednost[0] (*vrednost); vrednost[-1], vrednost[1]
- **int a[10];**
- deniše niz od 10 elemenata, tj. rezerviše 10 susednih objekata nazvanih a[0], a[1], . . . , a[9].
- Oznaka a[i] označava i-ti element niza.
- Ako je **pa** pokazivač na ceo broj, deklarisan sa
- **int *pa**;
- tada iskaz dodele
- pa = &a[0];
- dodeljuje promenljivoj pa pokazivač na nulti element niza a, tj. pa sadrži adresu
- elementa a[0]

Показивачи и низови





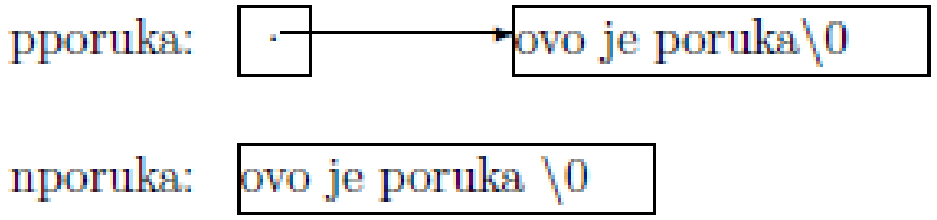
Показивачи и низови

- x = *pa;
- kopira sadržaj elementa a[0] u promenljivu x.
- Ako pokazivač **pa** pokazuje na neki određeni element niza, **pa+1** pokazuje na sledeći element a **pa-1** na prethodni element niza.
- Ako **pa** pokazuje na element a[0], onda je **pa+i** - adresa elementa a[i] a *(**pa+i**) je sadržaj elementa a[i]. Ova aritmetika nad pokazivačima (adresama) važi bez obzira na tip objekta - elementa niza.
- Ime niza **a** tj. izraz **a** ima **tip int [10]** a **vrednost adrese njegovog pocetnog** elementa; zato dodela
- **pa** = &a[0]; (pokazivač dobija adresu nultog elementa niza),
- ima ekvivalentno značenje kao i dodela
- **pa** = a;
- a tip **int [10]** se može konvertovati u tip **int ***.
- Zato f(int a[]) / f(int *)

Показивачи и низови

- a+i (adresa i-tog elementa) je isto što i &a[i],
- *(a+i) je isto što i a[i]
- ako je pa pokazivač, onda je *(pa+i) isto što i pa[i]
- razlika između pokazivača i imena niza:
  - pokazivač je promenljiva, pa ima smisla izraz pa=a ili pa++,
  - ime niza to nije, pa nema smisla izraz oblika a=pa ili a++
- Dakle, pokazivač pa jeste l-vrednost, ime niza to nije

- char nporuka[ ] = "ovo je poruka"; /* *niz* */
- char *pporuka = "ovo je poruka"; /* *pokazivac* */



Функције и показивачи на карактере

```
/* strlen: vraca duzinu stringa s */
int strlen(char *s)
{
    int n;
    for (n=0; *s != '\0'; s++)
            n++;
    return n;
}
```

- strlen(ptr); /* *char *ptr */,*
- strlen(array); /* *char array [100] */,*
- strlen("zdravo"); /* *niskovna konstanta */*

Функције и показивачи на карактере

```
/* strcpy: kopira t u s; verzija sa pokazivacima (1) */
char  *strcpy(char *s, const char *t)
{
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
    return s;
}
```

Функције и показивачи на карактере

```
/* strcpy: kopira t u s; verzija sa pokazivacima (2) */
char  *strcpy(char *s, const char *t)
{
    while ((*s++ = *t++) != '\0');
    return s;
}
/* strcpy: kopira t u s; verzija sa pokazivacima (3) */
char  *strcpy(char *s, const char *t)
{
    while (*s++ = *t++);
    return s;
}
```

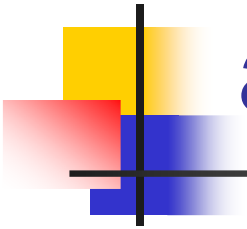
I Iоказивачи: дводимензиони низ и показивач на једнодимензиони низ

- Дводимензиони низ и показивач на једнодимензиони низ – сличности:
  - char a[10][100], (*b)[100];
  - b=a; (и a и b су показивачи на низ (врсту))
  - a[3][4], b[3][4] , *(*(b+3)+4)– обраћање једном карактеру
  - a[i][j], b[i][j] , *(*(b+i)+j)
- Разлике
  - Алокација меморије

Показивачи: дводимензиони низ и низ показивача

- Дводимензиони низ и низ показивача – сличности:
  - char a[10][100], *b[10];
  - b[i]=a[i];
  - a[3][4], b[3][4], *(*(b+3)+4) – обраћање једном карактеру
- Разлике:
  - Алокација меморије
  - char a[10][100]; простор од 1000 знакова; елементу a[i][j] додељен простор са индексом 100*i+j;
  - char *b[10]; простор само за 10 показивача
  - Обезбеђење меморије на коју ће ови показивачи да показују – експлицитно, функцијама malloc, calloc

Показивачи: функције алокације меморије



- **void *malloc(size_t n)**
- враћа показивач на **n** бајтова *неиницијализованог* простора или NULL ако простор не може да се додели
- **void *calloc(size_t n, size_t size)**
- враћа показивач на довољан простор у меморији за низ од **n** објеката наведене величине **size**, или **NULL**. Простор је *иницијализован са 0*.
- Тип **void * - некомплетни тип; недефинисана аритметика**
- повратни тип **void ***: повратна вредност (показивач) мора да буде придружен одговарајућем типу
- Пример:

```
int *ip;
ip=(int *)calloc(n, sizeof(int));
```

- Стандардно заглавље <stdlib.h>

Дводимензиони низови и низови показивача - примери

- b[i] = (char *)malloc(strlen(a[i])+1);

- показивач b[i] показује на простор величине стринга у i-тој врсти матрице a

- b[i] = (char *)malloc(sizeof(*b[i]));

- показивач b[i] показује на простор величине једног карактера

- b[i] = (char *)malloc(10*sizeof(*b[i]));

- показивач b[i] показује на простор величине 10 карактера – бајтова

- Оператор sizeof
  - унарни оператор – у време компилације
  - израчунава величину објекта или типа у бајтовима
  - враћа тип size_t – unsigned integer
  - дефинисан у стандардном заглављу <stddef.h>

Показивачи на функције

- у C-у функција није променљива, али
- може се дефинисати показивач на функцију,
- може му се доделити вредност
- могу се сместити у низ
- могу се проследити функцијама као аргументи
- могу се вратити као вредности функција

Показивачи на функције: пример – табелирање функције

- Функција *tabeliranje* за произвољну задату функцију израчунава и штампа њене вредности у задатом интервалу са задатим кораком

```
void tabeliranje(float (*f)(float), float a, float b, float h)
{
    float x=a;
    while(x<=b)
    {
        printf("%f, %f \n", x, (*f)(x));
        x+=h;
    }
}
```

Табелирање функције: програм

```
#include <stdio.h>
void tabeliranje(float (*f)(float), float, float, float);
float f1(float);

int main()
{
    tabeliranje(&f1, 1.0,2.0,0.01);
}

float f1(float x)
{
    return x*x-1;
}
```

Нула функције

```
#include <math.h>
#include <stdio.h>
#define EPS 0.0001
float nula(float a, float b, float (*f)(float));
float f1(float);
float f2(float);
main()
{
   float x,y;
   scanf("%f, %f", &x, &y);
   printf("%f, %f, %f\n", x,y, nula(x,y,&f1));
   scanf("%f, %f", &x, &y);
   printf("%f, %f, %f\n", x,y, nula(x,y,&f2));
}
```

Нула функције- половљење интервала

```
float nula(float a, float b, float (*f)(float))
{
   float x,z;
   int s;
   s=((*f)(a)<0);
   do {
      x=(a+b)/2.0;
      z=(*f)(x);
      if((z<0)==s) a=x; else b=x;
   } while (fabs(z)>EPS);
return x;
}
```

Нула функције

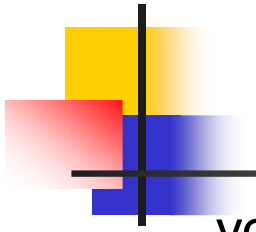
```
float f1(float x)
{
   float y;
   y= x - 1;
   return y;
}
float f2(float x)
{
   float y;
   y= x*x - 1;
   return y;
}
```

Стандардне функције bsearch и qsort

- Раније дефинисане функције претраживања и сортирања се примењују на низове са елементима одређеног типа (*int*, *char* и сл.)
- *Тип елемента низа одређује меморијски простор и адресу сваког елемента (адр(i+1.ел.) = адр(i.ел) + sizeof(tip))*
- Тип елемента одређује и начин поређења елемената (за *int, char* <, за стрингове функција *strcmp* итд).

- *У стандардном заглављу <stdlib.h> постоје, поред осталих, генеричке (рекурзивне) функције за брзо сортирање низа и бинарно претраживање сортираног низа са елементима произвољног типа, qsort и bsearch.*
- Потребно је да се прецизира величина меморијског простора за један елемент низа, и функција којом се врши поређење елемената
- Функција поређења се преноси као аргумент

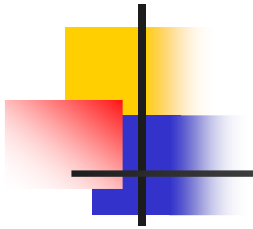
Стандардна функција qsort - декларација



void qsort(void *base, size_t n, size_t size, int (*cmp)(const void *, const void *)

- base – име низа који се сортира
- n – број елемената низа
- size – величина елемента низа (у бајтовима)
- cmp – показивач на функцију поређења која
  - има два аргумента - показивача на елементе низа
  - враћа int:
    - 0 ако су два елемента једнака
    - < 0 ако је вредност првог елемента "мања" од (испред) вредности другог
    - > 0 ако је вредност првог елемента "већа" од (иза) вредности другог
- Функција qsort, као ни раније функције сортирања, не враћа вредност

Стандардна функција bsearch - декларација



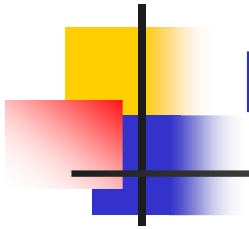
- void * bsearch (const void * key, const void *base, size_t n, size_t size, int (*cmp)(const void * , const void *)

- base, n, size – као и за функцију qsort
- key – показивач на вредност која се тражи у низу base
- Функција cmp пореди вредност кључа са појединачним елементом низа base и има облик

```
int cmp (const void * key, const void * pelem);
```

  - Први аргумент је показивач на вредност која се тражи, а други – показивач на елемент низа у коме се тражи
  - Типови аргумената не морају да буду исти
- Функција bsearch враћа показивач на тип елемента низа или NULL
- **Niz mora da bude sortiran!!!**

Функције линеарног претраживања: lfind, lsearch



- void *lfind(const void *key, const void *base, size_t num, size_t size, int (*fcmp)(const void *, const void*));
- void *lsearch(const void *key, void *base, size_t *num,size_t size, int (*fcmp)(const void *, const void *));
- Низ не мора да буде сортиран

Стандардна функција qsort: Сортирање целобројног низа

```
#include <stdio.h>
#include <stdlib.h>
#define NMAX 10
int v[NMAX]={0,1, -1, 2, -2, 3, -3, 4, -4, 5};
int cmpint(const void *, const void *);
int main()
{
    int i;
    qsort(v, NMAX, sizeof(int), &cmpint);
    for(i=0;i<NMAX;i++) printf("%d\n", v[i]);
    return 0;
}
int cmpint(const void * x, const void * y)
{       return *(int *)x - *(int*)y;  }
```

Стандардна функција qsort: Сортирање низа карактера

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char ch[]= "Ovo je niz karaktera ne duzi od 40";
int chcmp(const void *, const void *);
int main()
{   int i;
    qsort(ch, strlen(ch), sizeof(char), &chcmp);
    printf("%s\n", ch);
    return 0;
}
int chcmp(const void * x, const void * y)
{       return *(char *) x - *(char *)y;  }
```

Стандардна функција qsort:

Сортирање низа показивача на стрингове – лексикографски

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NREC 4
char *reci[]={"auto", "while", "break", "do"};
int cmpreci(const  void *, const void *);
int main(){
        int i;
        qsort(reci, NREC, sizeof(char *), &cmpreci);
        for (i=0;i<NREC;i++)
             printf("%s\n", reci[i]);
        return 0;
}
int cmpreci(const void * a, const void * b){
        const char ** s1=(const char **)a;
        const char ** s2=(const char **)b;
        return strcmp(*s1, *s2);
}
```

Стандардна функција qsort:

Сортирање низа показивача на стрингове по дужини

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NREC 4

char *reci[]={"auto",  "break", "while", "do"};
int cmprecid(const  void *, const void *);
int main() {
        int i;
        qsort(reci, NREC, sizeof(char *), &cmprecid);
        for (i=0;i<NREC;i++)
             printf("%s\n", reci[i]);
        return 0; }
int cmprecid(const void * a, const void * b) {
        const char ** s1= (const char **)a;
        const char ** s2=(const char **)b;
        return strlen(*s1)-strlen(*s2);
}
```

Стандардна функција qsort:

Сортирање низа показивача на стрингове по дужини, за исте дужине лексикографски

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NREC 4

char *reci[]={"auto",  "break", "while", "do"};
int cmprecid(const  void *, const void *);
int main() {
        int i;
        qsort(reci, NREC, sizeof(char *), &cmprecid);
        for (i=0;i<NREC;i++)
             printf("%s\n", reci[i]);
        return 0; }
int cmprecid(const void * a, const void * b) {
        const char ** s1= (const char **)a;
        const char ** s2=(const char **)b;
        if (strlen(*s1)==strlen(*s2)) return strcmp(*s1, *s2);
           else return strlen(*s1)-strlen(*s2);
}
```

Стандардна функција qsort: Сортирање низа структура

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NKLJUC 4
struct kljuc {
        char * rec;
        int brojac;
} kljucreci[NKLJUC] = {
            "auto" , 0, "while", 0, "break", 0, "do", 0};
int cmpstruct(const void *, const void *);
int main() {
    int i;
    qsort(kljucreci, NKLJUC, sizeof(struct kljuc), &cmpstruct);
    for(i=0;i<NKLJUC;i++) printf("%s\n", kljucreci[i]);
    return 0; }
int cmpstruct(const void *s1, const void *s2)
{    return strcmp(((struct kljuc *) s1)->rec, ((struct kljuc *)s2)->rec);  }
```

Стандардна функција bsearch :Претраживање низа целих бројева

Niz mora da bude sortiran!!!

```
#include <stdio.h>
#include <stdlib.h>
int cmpint(const void *, const void *);
int main()
{    int i, *p, broj, nizbrojeva[]={1,2,3,4,5,6,7,8,9,10};
   scanf("%d", &broj);
   printf("%d\n", broj);
   for(i=0;i<10;i++) printf("%d\n", nizbrojeva[i]);
   p = (int *)bsearch(&broj, nizbrojeva, 10, sizeof(int), &cmpint);
   if(p!=NULL) printf(" %d\n", *p);
   return 0;
}
```

(cmpint као у примеру за сортирње)

Стандардна функција bsearch : Претраживање низа структура

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NKLJUC 4
struct kljuc {
        char rec[10];
        int brojac;
} *p, kljucreci[NKLJUC] = {"auto" , 0, "break", 0, "do", 0, "while", 0};
int strtcmp(const void *, const void *);
int main()
{
   int i;
   char krec[10];
   scanf("%s", krec);
   p = (struct kljuc *)bsearch(krec, kljucreci, NKLJUC, sizeof(struct kljuc), &strtcmp);
   if(p!=NULL)printf("%s, %d\n", p->rec, p->brojac);
   return 0;
}
int strtcmp(const void * key, const void * entry)
{ return strcmp((char *)key, ((struct kljuc *)entry)->rec); }
```