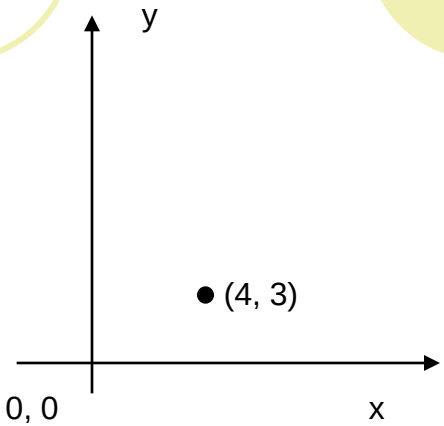
vladaf@matf.bg.ac.rs

Strukture

vladaf@matf.bg.ac.rs

6.1 Osnove struktura

```
struct oznaka {
    tip imeprom;
    ...
};
```




```
struct point {
    int x;
    int y;
};
```

```
struct { … } x, y, z;
```

```
int x, y, z;
```

```
struct point pt;
```

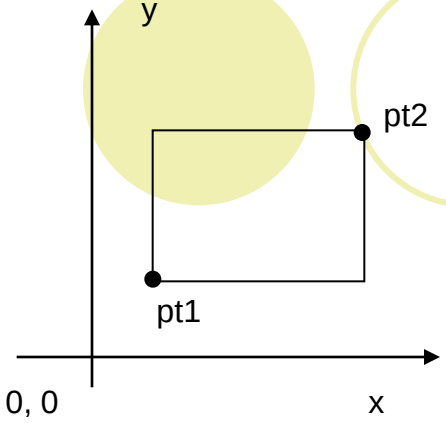
```
struct point maxpt = { 320, 200 };
```

ime-strukture.član

```
printf("%d,%d", pt.x, pt.y);
```

```
double dist, sqrt(double);
dist = sqrt((double)pt.x * pt.x + (double)pt.y * pt.y);
```

vladaf@matf.bg.ac.rs




```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

```
struct rect screen;
```

```
screen.pt1.x;
```

vladaf@matf.bg.ac.rs

6.2 Strukture i funkcije

```
/* makepoint:  make a point from x and y components */
struct point makepoint(int x, int y)
{
    struct point temp;

    temp.x = x;
    temp.y = y;
    return temp;
}
```

```
struct rect screen;
struct point middle;
struct point makepoint(int, int);

screen.pt1 = makepoint(0, 0);
screen.pt2 = makepoint(XMAX, YMAX);
middle = makepoint((screen.pt1.x + screen.pt2.x)/2,
                   (screen.pt1.y + screen.pt2.y)/2);
```

```
/* addpoint:  add two points */
struct point addpoint(struct point p1, struct point p2)
{
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
```

vladaf@matf.bg.ac.rs

```
/* ptinrect:  return 1 if p in r, 0 if not */
int ptinrect(struct point p, struct rect r)
{
    return p.x >= r.pt1.x && p.x < r.pt2.x
        && p.y >= r.pt1.y && p.y < r.pt2.y;
}
```

```
#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

/* canonrect:  canonicalize coordinates of rectangle */
struct rect canonrect(struct rect r)
{
    struct rect temp;

    temp.pt1.x = min(r.pt1.x, r.pt2.x);
    temp.pt1.y = min(r.pt1.y, r.pt2.y);
    temp.pt2.x = max(r.pt1.x, r.pt2.x);
    temp.pt2.y = max(r.pt1.y, r.pt2.y);
    return temp;
}
```

```
struct point *pp;
```

```
(*pp).x
```

```
(*pp).y
```

```
struct point origin, *pp;
pp = &origin;
printf("origin is (%d, %d) \n", (*pp).x, (*pp).y);
```

p->član-strukture

```
printf("origin is (%d, %d) \n", pp->x, pp->y);
```

vladaf@matf.bg.ac.rs

```
struct rect r, *rp = &r;
```

```
r.pt1.x
rp->pt1.x
(r.pt1).x
(rp->pt1).x
```

```
() [] -> .
```

```
struct {
    int len;
    char *str;
} *p;
```

```
++p->len
```
uvecava len

```
(++p)->len
```
uvecava p

```
(p++)->len
```
uvecava p nakon pristupanja clanu len, zagrade nisu neophodne

```
*p->str       /* string na koji pokazivac str pokazuje */
*p->str++    /* uvecava pokazivac str nakon pristupanja onome na sta on pokazuje, slicno kao *s++ */
(*p->str)++ /* uvecava ono nasta str pokazuje */
*p++->str    /* uvecava p nakon pristupanja onome nasta str pokazuje */
```

vladaf@matf.bg.ac.rs

6.3 Nizovi struktura

Primer: brojanje primeraka svake ključne reči iz jezika C

```
char *keyword[NKEYS];
int keycount[NKEYS];
```

```
char *word;
int count;
```

```
struct key {
    char *word;
    int count;
} keytab[NKEYS];
```

```
struct key {
    char *word;
    int count;
} ;

struct key keytab[NKEYS];
```

vladaf@matf.bg.ac.rs

```
struct key {
    char *word;
    int count;
} keytab[] = {
    "auto", 0,
    "break", 0,
    "case", 0,
    "char", 0,
    "const", 0,
    "continue", 0,
    "default", 0,
    /* ... */
    "unsigned", 0,
    "void", 0,
    "volatile", 0,
    "while", 0
};
```

```
    { "auto", 0 },
    { "break", 0 },
    { "case", 0 },
    …
```

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

#define MAXWORD 100

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

/* count C keywords */
main()
{
    int n;
    char word[MAXWORD];

    while (getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            if ((n = binsearch(word, keytab, NKEYS)) >= 0)
                keytab[n].count++;
    for (n = 0; n < NKEYS; n++)
        if (keytab[n].count > 0)
            printf("%4d %s\n",
                keytab[n].count, keytab[n].word);
    return 0;
}
/* binsearch:  find word in tab[0]...tab[n-1] */
int binsearch(char *word, struct key tab[], int n)
{
    int cond;
    int low, high, mid;

    low = 0;
    high = n - 1;
    while (low <= high) {
        mid = (low+high) / 2;
        if ((cond = strcmp(word, tab[mid].word)) < 0)
            high = mid - 1;
        else if (cond > 0)
            low = mid + 1;
        else
            return mid;
    }
    return -1;
}
```

vladaf@matf.bg.ac.rs

```
#define NKEYS (sizeof keytab / sizeof(struct key))
```

```
#define NKEYS (sizeof keytab / sizeof keytab[0])
```

```
/* getword:  get next word or character from input */
int getword(char *word, int lim)
{
    int c, getch(void);
    void ungetch(int);
    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];
}
```

6.4 Pokazivači na strukture

vladaf@matf.bg.ac.rs

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAXWORD 100

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

/* count C keywords; pointer version */
main()
{
    char word[MAXWORD];
    struct key *p;

    while (getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            if ((p=binsearch(word, keytab, NKEYS)) != NULL)
                p->count++;
    for (p = keytab; p < keytab + NKEYS; p++)
        if (p->count > 0)
            printf("%4d %s\n", p->count, p->word);
    return 0;
}

/* binsearch: find word in tab[0]...tab[n-1] */
struct key *binsearch(char *word, struct key *tab, int n)
{
    int cond;
    struct key *low = &tab[0];
    struct key *high = &tab[n];
    struct key *mid;

    while (low < high) {
        mid = low + (high-low) / 2;
        if ((cond = strcmp(word, mid->word)) < 0)
            high = mid;
        else if (cond > 0)
            low = mid + 1;
        else
            return mid;
    }
    return NULL;
}
```

```
for (p = keytab; p < keytab + NKEYS; p++)
```

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

8 bajtova

sizeof

```
struct key *binsearch(char *word, struct key *tab, int n)
```

```
struct key *
binsearch(char *word, struct key *tab, int n)
```

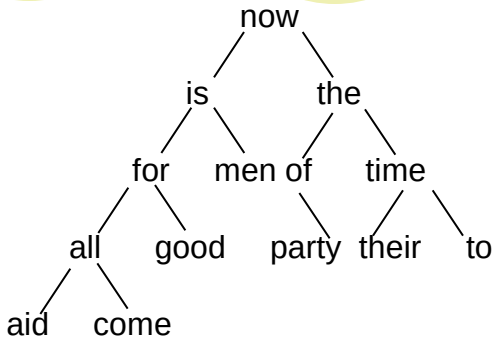
```
mid = low + high / 2
/* pogrešno */
```

vladaf@matf.bg.ac.rs

6.5 Samoreferencirajuće strukture

Rešiti opštiji problem brojanja primeraka svih reči na ulazu

now is the time for all good men to come to the aid of their party



Struktura podataka: binarno stablo sadrži po jedan "čvor" za svaku različitu reč, svaki čvor sadrži:

- pokazivač na tekst reči
- broj primeraka te reči
- pokazivač na levi čvor dete
- pokazivač na desni čvor dete

```
struct tnode {              /* the tree node: */
    char *word;             /* points to the text */
    int count;              /* number of occurrences */
    struct tnode *left;     /* left child */
    struct tnode *right;    /* right child */
};
```

```
struct tnode *left;
```

```
struct t {
    …
    struct s *p;   /* p pokazuje na s */
};
struct s {
    …
    struct t *q;   /* q pokazuje na t */
};
```

vladaf@matf.bg.ac.rs

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

#define MAXWORD 100
struct tnode *addtree(struct tnode *, char *);
void treeprint(struct tnode *);
int getword(char *, int);

/* word frequency count */
main()
{
    struct tnode *root;
    char word[MAXWORD];

    root = NULL;
    while (getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            root = addtree(root, word);
    treeprint(root);
    return 0;
}
```

```
struct tnode *talloc(void);
char *strdup(char *);
/* addtree:  add a node with w, at or below p */
struct treenode *addtree(struct tnode *p, char *w)
{
    int cond;

    if (p == NULL) {          /* a new word has arrived */
        p = talloc();         /* make a new node */
        p->word = strdup(w);
        p->count = 1;
        p->left = p->right = NULL;
    } else if ((cond = strcmp(w, p->word)) == 0)
        p->count++;           /* repeated word */
    else if (cond < 0)        /* less than into left subtree */
        p->left = addtree(p->left, w);
    else                      /* greater than into right subtree */
        p->right = addtree(p->right, w);
    return p;
}
```

vladaf@matf.bg.ac.rs

```
/* treeprint:  in-order print of tree p */
void treeprint(struct tnode *p)
{
    if (p != NULL) {
        treeprint(p->left);
        printf("%4d %s\n", p->count, p->word);
        treeprint(p->right);
    }
}
```

```
#include <stdlib.h>
/* talloc:  make a tnode */
struct tnode *talloc(void)
{
    return (struct tnode *) malloc(sizeof(struct tnode));
}
```

```
char *strdup(char *s)                    /* make a duplicate of s */
{
    char *p;

    p = (char *) malloc(strlen(s)+1);    /* +1 for '\0' */
    if (p != NULL)
        strcpy(p, s);
    return p;
}
```

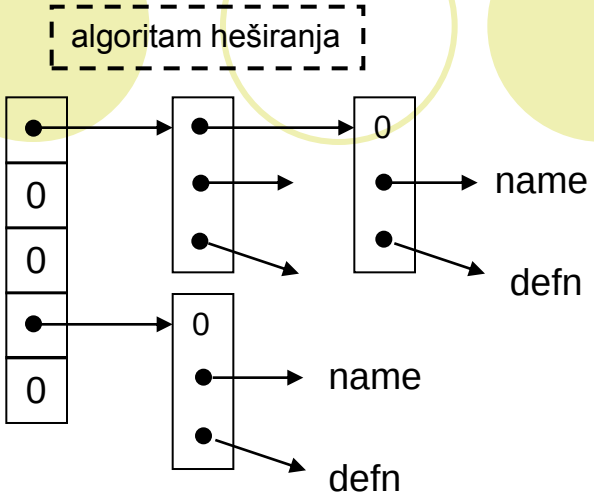
vladaf@matf.bg.ac.rs

6.6 Pretraživanje tabele

```
#define IN 1
```

```
state = IN;
```

```
install(s, t)
lookup(s)
```




```
struct nlist {              /* table entry: */
    struct nlist *next;     /* next entry in chain */
    char *name;             /* defined name */
    char *defn;             /* replacement text */
};
```

```
#define HASHSIZE 101
static struct nlist *hashtab[HASHSIZE];  /* pointer table */
```

```
/* hash:  form hash value for string s */
unsigned hash(char *s)
{
    unsigned hashval;

    for (hashval = 0; *s != '\0'; s++)
        hashval = *s + 31 * hashval;
    return hashval % HASHSIZE;
}
```

vladaf@matf.bg.ac.rs

```
/* lookup:  look for s in hashtab */
struct nlist *lookup(char *s)
{
    struct nlist *np;

    for (np = hashtab[hash(s)];  np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;          /* found */
    return NULL;                /* not found */
}
```

```
for (ptr = head; ptr != NULL; ptr = ptr->next)
                …
```

```
struct nlist *lookup(char *);
char *strdup(char *);

/* install:  put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn)
{
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    } else       /* already there */
        free((void *) np->defn);   /*free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```

vladaf@matf.bg.ac.rs

6.7 Typedef

```
typedef int Length;
```

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

```
typedef char *String;
```

```
String p, lineptr[MAXLINES], alloc(int);
int strcmp(String, String);
p = (String) malloc(100);
```

```
typedef struct tnode *Treeptr;

typedef struct tnode { /* the tree node: */
    char *word;          /* points to the text */
    int count;           /* number of occurrences */
    struct tnode *left;  /* left child */
    struct tnode *right; /* right child */
} Treenode;
```

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

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

```
PFI strcmp, numcmp
```

vladaf@matf.bg.ac.rs

6.8 Unije

```
union u_tag {
        inti ival;
        float fval;
        char *sval;
} u;
```

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

```
if (utype == INT)
    printf("%d\n", u.ival);
else if (utype == FLOAT)
    printf("%f\n", u.fval);
else if (utype == STRING)
    printf("%s\n", u.sval);
else
    printf("bad type %d in utype\n", utype);
```

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

```
symtab[i].u.ival
```

```
*symtab[i].u.sval  ⟷  symtab[i].u.sval[0]
```

vladaf@matf.bg.ac.rs

6.9 Bit-polja

```
#define KEYWORD  01
#define EXTERNAL 02
#define STATIC   04
```

```
enum { KEYWORD = 01, EXTERNAL = 02, STATIC = 04 };
```

```
flags |= EXTERNAL | STATIC;
```

```
flags &= ~(EXTERNAL | STATIC);
```

```
if ((flags & (EXTERNAL | STATIC)) == 0) …
```

```
struct {
    unsigned int is_keyword : 1;
    unsigned int is_extern  : 1;
    unsigned int is_static  : 1;
} flags;
```

```
flags.is_extern = flags.is_ static = 1;
```

```
flags.is_extern = flags.is_ static = 0;
```

```
if (flags.is_extern == 0 && flags.is_ static == 0)
                …
```

ne može se primeniti operator &