

Pokazivači i nizovi

5.1 Pokazivači i adrese

unarni operatori:

& (adresa)

* (posredni pristup)

`p = &c;`

`int x = 1, y = 2, z[10];`

`int *ip;` */* ip je pokazivač na int */*

`ip = &x;` */* ip sada pokazuje na x, odnosno promenljivoj ip dodeljuje se adresa od x */*

`y = *ip;` */* y je sada 1 */*

`*ip = 0;` */* x je sada 0 */*

`ip = &z[0];` */* ip sada pokazuje na z[0] */*

`int *ip;`

`double *dp, atof(char *);`

`*ip = *ip + 10;`

`y = *ip + 1`

`*ip += 1`

`++*ip`

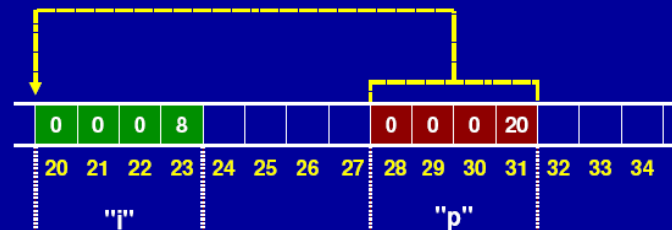
`(*ip)++`

`iq = ip`

Introduction to C programming

21

Pointers (4)



`int i = 8`
`&i = 20`

`int *p = &i (=20)`
`&p = 28`
`int j = *p (= 8)`

5.2 Pokazivači i argumenti funkcija

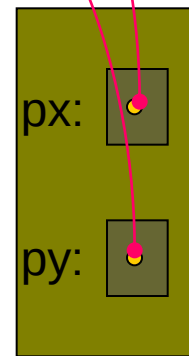
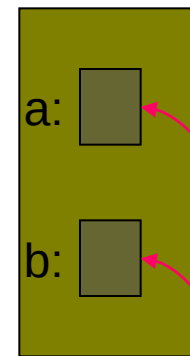
```
swap(a, b);
```

```
void swap(int x, int y) /* POGREŠNO */  
{  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
}
```

```
swap(&a, &b);
```

```
void swap(int *px, int *py) /* interchange *px and *py */  
{  
    int temp;  
    temp = *px;  
    *px = *py;  
    *py = temp;  
}
```

u pozivaocu



u funkciji swap

```
int n, array[SIZE], getint(int *);  
for (n = 0; n < SIZE && getint(&array[n]) != EOF; n++)  
    ;
```

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

```
int getch(void);  
void ungetch(int);
```

```
/* getint: get next integer from input into *pn */  
int getint(int *pn)  
{  
    int c, sign;  
  
    while (isspace(c = getch())) /* skip white space */  
        ;  
    if (!isdigit(c) && c != EOF && c != '+' && c != '-') {  
        ungetch(c); /* it is not a number */  
        return 0;  
    }  
    sign = (c == '-') ? -1 : 1;  
    if (c == '+' || c == '-')  
        c = getch();  
    for (*pn = 0; isdigit(c); c = getch())  
        *pn = 10 * *pn + (c - '0');  
    *pn *= sign;  
    if (c != EOF)  
        ungetch(c);  
    return c;  
}
```

5.3 Pokazivači i nizovi

```
int a[10];
```



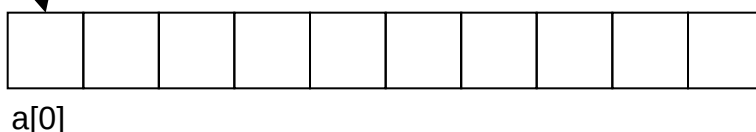
```
int *pa;
```

pa:



```
pa = &a[0];
```

a:



```
x = *pa; /* kopira se sadržaj elementa a[0] u x */
```

```
*(pa+1)
```

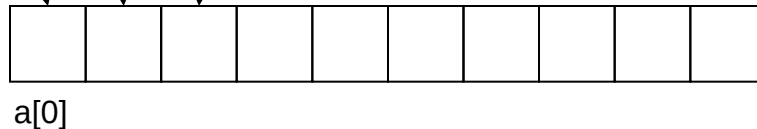
pa:



pa+1:

pa+2:

a:



```
pa = &a[0];
```

```
pa = a;
```

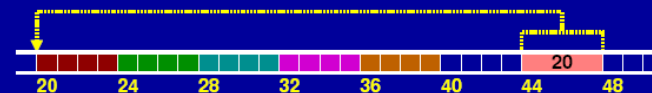
$a[i] \longleftrightarrow *(a+i)$

$\& a[i] \longleftrightarrow \& *(a+i)$
 $\& a[i] \longleftrightarrow a+i$

Introduction to C programming

45

Pointer Arithmetic (4)



```
int array[5];  
int *p = array;
```

ako je pa pokazivač =>

$pa[i] \longleftrightarrow *(pa+i)$ =>

(niz i indeks) $\longleftrightarrow$ (pokazivač i relativno odstojanje)

```
/* strlen: return length of string s */  
int strlen(char *s)  
{  
    int n;  
    for (n = 0; *s != '\0', s++)  
        n++;  
    return n;  
}
```

strlen("hello, world");	/* string konstanta */
strlen(array);	/* char array[100]; */
strlen(ptr);	/* char *ptr; */

char s[]; ↔ char *s;

f(&a[2]) ↔ f(a+2)

f(int arr[]) { ... } ↔ f(int *arr) { ... }

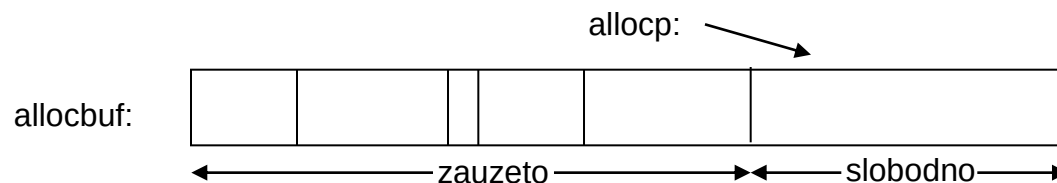
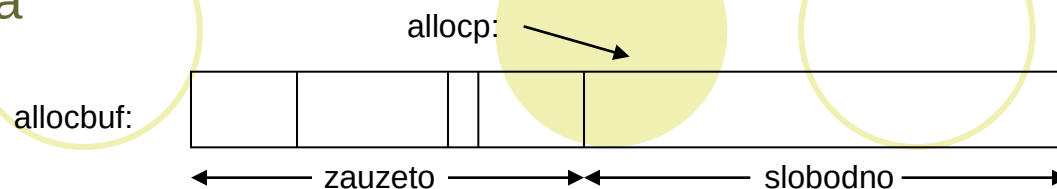
5.4 Adresna aritmetika

`alloc(n)`

`afree(p)`

`malloc`

`free`



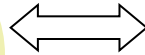
```
#define ALLOCSIZE 10000                /* size of available space */

static char allocbuf[ALLOCSIZE];       /* storage for alloc */
static char *allocp = allocbuf;        /* next free position */

char *alloc(int n)                     /* return pointer to n characters */
{
    if (allocbuf + ALLOCSIZE - allocp >= n) { /* it fits */
        allocp += n;
        return allocp - n;              /* old p */
    } else                               /* not enough room */
        return 0;
}

void afree(char *p)                    /* free storage pointed to by p */
{
    if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
        allocp = p;
}
```

```
static char *allocp = allocbuf;
```



```
static char *allocp = &allocbuf[0];
```

```
if (allocbuf + ALLOCSIZE - allocp >= n) { /* it fits */
```

```
NULL <stddef.h> (stdio.h i stdlib.h)
```

```
if (allocbuf + ALLOCSIZE - allocp >= n) { /* it fits */
```

```
if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
```

```
p < q
```

```
p+n
```

```
p-q (broj elemenata od p do q)
```

```
/* strlen: return length of string s */
```

```
int strlen(char *s)
```

```
{
```

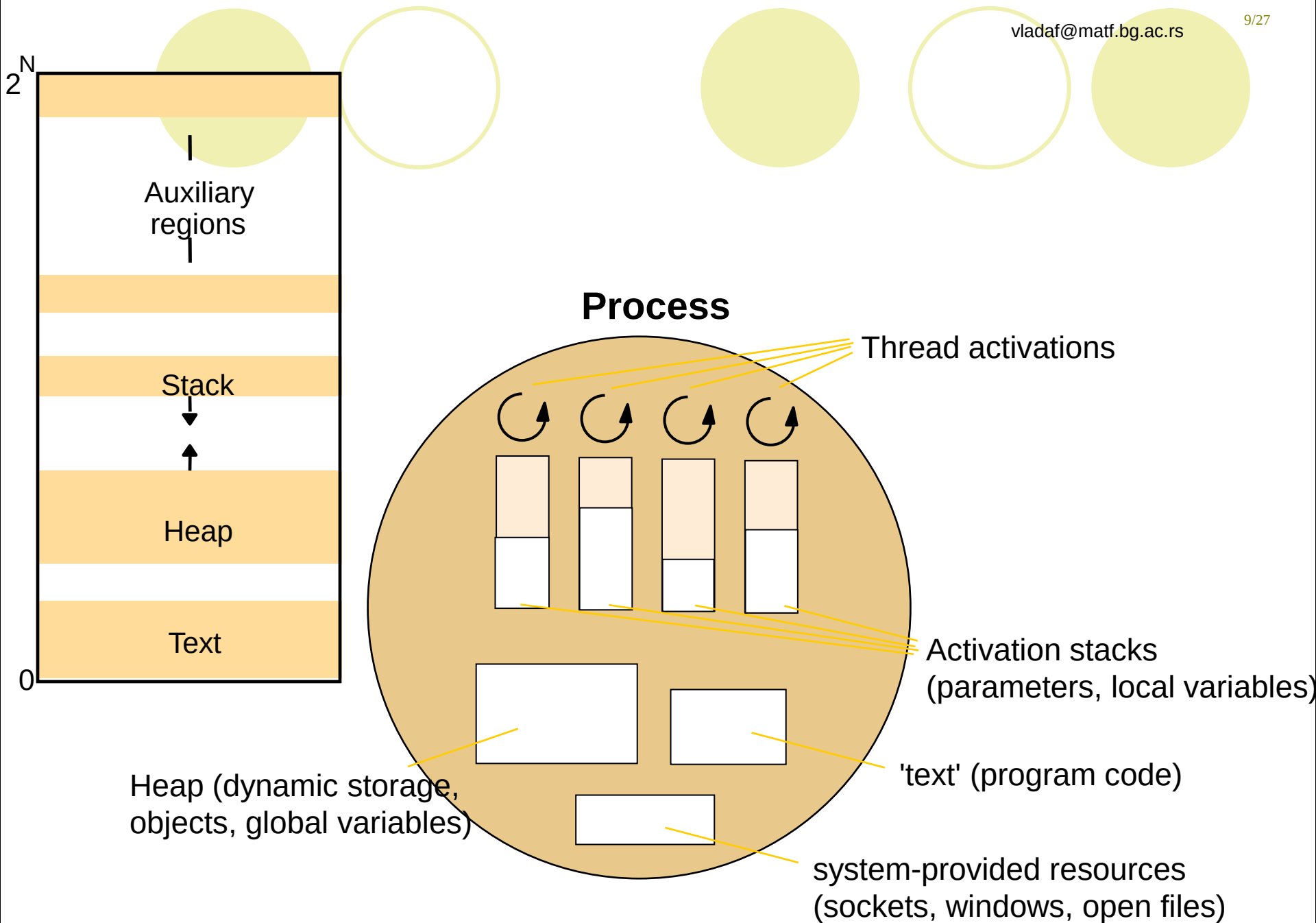
```
    char *p = s;
```

```
    while (*p != '\0')
```

```
        p++;
```

```
    return p - s;
```

```
}
```



5.5 Znakovni pokazivači i funkcije

```
"Ja sam string"
```

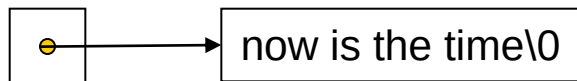
```
printf("hello, world\n");
```

```
char *pmessage;
```

```
pmessage = "now is the time";
```

```
char amessage[ ] = "now is the time"; /* niz */  
char *pmessage = "now is the time"; /* pokazivač */
```

pmessage:



amessage:

now is the time\0

```

/* strcpy: copy t to s; array subscript version */
void strcpy(char *s, char *t)
{
    int i;

    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}

```

```

/* strcpy: copy t to s; pointer version 1 */
void strcpy(char *s, char *t)
{
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}

```

```

/* strcpy: copy t to s; pointer version 2 */
void strcpy(char *s, char *t)
{
    while ((*s++ = *t++) != '\0')
        ;
}

```

```

/* strcpy: copy t to s; pointer version 3 */
void strcpy(char *s, char *t)
{
    while (*s++ = *t++)
        ;
}

```

```

/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
    int i;

    for (i = 0; s[i] == t[i]; i++)
        if (s[i] == '\0')
            return 0;
    return s[i] - t[i];
}

```

```

/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
    for ( ; *s == *t; s++, t++)
        if (*s == '\0')
            return 0;
    return *s - *t;
}

```

```

*--p

```

```

*p++ = val; /* push dodati val na stek */
val = *--p; /* pop skinuti vrh steka u val */

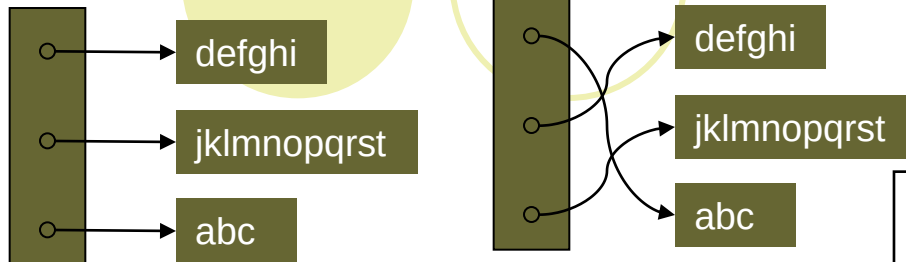
```

```

<string.h>

```

5.6 Nizovi pokazivača; pokazivači na pokazivače



pročitati sve redove ulaznog teksta
sortirati ih
prikazati ih po redu

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

#define MAXLINES 5000      /* max #lines to be sorted */

char *lineptr[MAXLINES]; /* pointers to text lines */

int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);

void qsort(char *lineptr[], int left, int right);

/* sort input lines */
main()
{
    int nlines; /* number of input lines read */

    if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
        qsort(lineptr, 0, nlines-1);
        writelines(lineptr, nlines);
        return 0;
    } else {
        printf("error: input too big to sort\n");
        return 1;
    }
}
```

```
#define MAXLEN 1000 /* max length of any input line */
int getline(char *, int);
char *alloc(int);

/* readlines: read input lines */
int readlines(char *lineptr[], int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];

    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
        if (nlines >= maxlines || (p = alloc(len)) == NULL)
            return -1;
        else {
            line[len-1] = '\0'; /* delete newline */
            strcpy(p, line);
            lineptr[nlines++] = p;
        }
    return nlines;
}

/* writelines: write output lines */
void writelines(char *lineptr[], int nlines)
{
    int i;

    for (i = 0; i < nlines; i++)
        printf("%s\n", lineptr[i]);
}
```

```
/* writelines: write output lines */
void writelines(char *lineptr[ ], int nlines)
{
    while (nlines-- > 0)
        printf("%s\n", *lineptr++);
}
```

```
/* qsort: sort v[left]...v[right] into increasing order */
void qsort(char *v[ ], int left, int right)
{
    int i, last;
    void swap(char *v[ ], int i, int j);

    if (left >= right) /* do nothing if array contains */
        return;      /* fewer than two elements */
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
        if (strcmp(v[i], v[left]) < 0)
            swap(v, ++last, i);
    swap(v, left, last);
    qsort(v, left, last-1);
    qsort(v, last+1, right);
}
```

```
/* swap: interchange v[i] and v[j] */
void swap(char *v[ ], int i, int j)
{
    char *temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```

5.7 Višedimenzionalni nizovi

```
month_day(1998, 60, &m, &d)
```

```
daytab[i][j] /* [red][kolona] */
```

```
daytab[i, j] /* POGREŠNO */
```

```
f(int daytab[2][13]) { ... }
```

```
f(int daytab[ ][13]) { ... }
```

```
f(int (*daytab)[13]) { ... }
```

```
int *daytab[13]
/* ovo je niz od 13 pokazivača */
```

```
static char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

/* day_of_year: set day of year from month & day */
int day_of_year(int year, int month, int day)
{
    int i, leap;

    leap = year%4 == 0 && year%100 != 0 || year%400 == 0;
    for (i = 1; i < month; i++)
        day += daytab[leap][i];
    return day;
}

/* month_day: set month, day from day of year */
void month_day(int year, int yearday, int *pmonth, int *pday)
{
    int i, leap;

    leap = year%4 == 0 && year%100 != 0 || year%400 == 0;
    for (i = 1; yearday > daytab[leap][i]; i++)
        yearday -= daytab[leap][i];
    *pmonth = i;
    *pday = yearday;
}
```

5.8 Inicijalizovanje pokazivačkih nizova

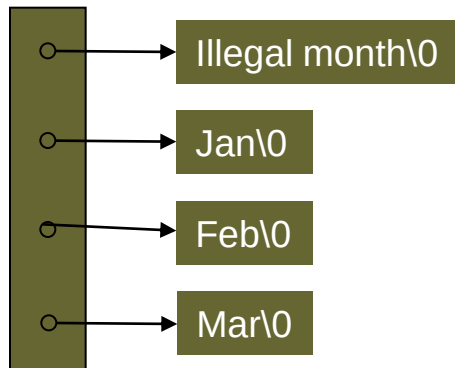
```
/* month_name: return name of n-th month */  
char *month_name(int n)  
{  
    static char *name[ ] = {  
        "Illegal month",  
        "January", "February", "March",  
        "April", "May", "June",  
        "July", "August", "September",  
        "October", "November", "December"  
    };  
    return (n < 1 || n > 12) ? name[0] : name[n];  
}
```

5.9 Pokazivači i višedimenzionalni nizovi

```
int a[10][20];      /* dvodimenzionalni niz */  
int *b[10];         /* niz pokazivača */
```

```
char *name[] = { "Illegal month", "Jan", "Feb", "Mar" };
```

name:



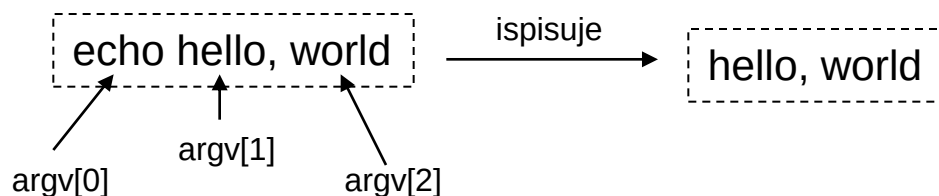
```
char aname[ ][15] = { "Illegal month", "Jan", "Feb", "Mar" };
```

aname:

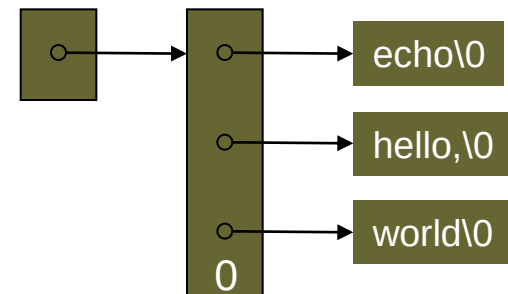


5.10 Argumenti komandne linije

argc (argument count), argv (argument vector)



argv:



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

```
/* echo command-line arguments; 1st version */
```

```
main(int argc, char *argv[ ])
```

```
{
```

```
    int i;
```

```
    for (i = 1; i < argc; i++)
```

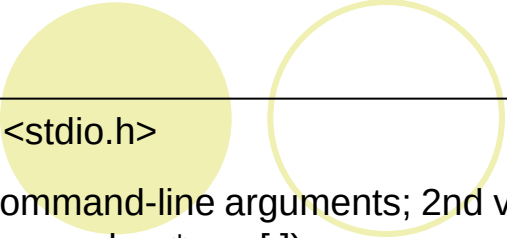
```
        printf("%s%s", argv[i], (i < argc-1) ? " " : "");
```

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

```
    return 0;
```

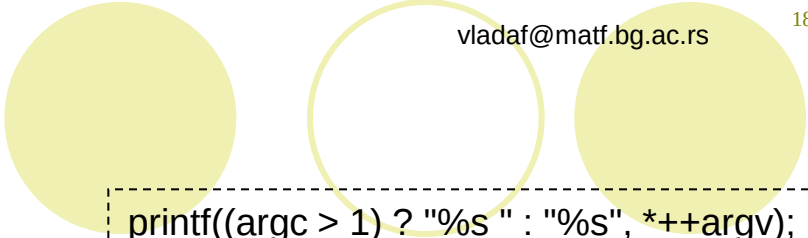
```
}
```

`argv[argc]` nulti pokazivač



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

```
/* echo command-line arguments; 2nd version */
main(int argc, char *argv[ ])
{
    while (--argc > 0)
        printf("%s%s", *++argv, (argc > 1) ? " " : "");
    printf("\n");
    return 0;
}
```



```
printf((argc > 1) ? "%s " : "%s", *++argv);
```

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

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

```
#define MAXLINE 1000
```

```
int getline(char *line, int max);
```

```
/* find: print lines that match pattern from 1st arg */
```

```
main(int argc, char *argv[])
```

```
{
```

```
    char line[MAXLINE];
```

```
    int found = 0;
```

```
    if (argc != 2)
```

```
        printf("Usage: find pattern\n");
```

```
    else
```

```
        while (getline(line, MAXLINE) > 0)
```

```
            if (strstr(line, argv[1]) != NULL) {
```

```
                printf("%s", line);
```

```
                found++;
```

```
            }
```

```
    return found;
```

```
}
```

find -x -n *obrazac*

```
#include <stdio.h>
#include <string.h>
#define MAXLINE 1000
int getline(char *line, int max);
/* find: print lines that match pattern from 1st arg */
main(int argc, char *argv[])
{
    char line[MAXLINE];
    long lineno = 0;
    int c, except = 0, number = 0, found = 0;
    while (--argc > 0 && (*++argv)[0] == '-')
        while (c = *++argv[0])
            switch (c) {
                case 'x':
                    except = 1;
                    break;
                case 'n':
                    number = 1;
                    break;
                default:
                    printf("find: illegal option %c\n", c);
                    argc = 0;
                    found = -1;
                    break;
            }
    if (argc != 1)
        printf("Usage: find -x -n pattern\n");
    else
        while (getline(line, MAXLINE) > 0) {
            lineno++;
            if ((strstr(line, *argv) != NULL) != except) {
                if (number)
                    printf("%ld:", lineno);
                printf("%s", line);
                found++;
            }
        }
    return found;
}
```

Uvod u programiranje

vladaf@matf.bg.ac.rs

19/27

(*++argv)[0] /* prvi znak niza
elemenata */

++argv[0] / specifican
string argumenta */

5.11 Pokazivači na funkcije

program sortira ulazne redove u
numerickom a ne leksikografskom
redosledu (opcionim argument -n)

strcmp
numcmp

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

#define MAXLINES 5000          /* max #lines to be sorted */
char *lineptr[MAXLINES];      /* pointers to text lines */

int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);

void qsort(void *lineptr[], int left, int right,
            int (*comp)(void *, void *));
int numcmp(char *, char *);

/* sort input lines */
main(int argc, char *argv[])
{
    int nlines;                /* number of input lines read */
    int numeric = 0;           /* 1 if numeric sort */

    if (argc > 1 && strcmp(argv[1], "-n") == 0)
        numeric = 1;
    if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
        qsort((void**) lineptr, 0, nlines-1,
              (int (*)(void*,void*))(numeric ? numcmp : strcmp));
        writelines(lineptr, nlines);
        return 0;
    } else {
        printf("input too big to sort\n");
        return 1;
    }
}
```

```
/* qsort: sort v[left]...v[right] into increasing order */
void qsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
    int i, last;
    void swap(void *v[], int, int);
    if (left >= right) /* do nothing if array contains */
        return;      /* fewer than two elements */
    swap(v, left, (left + right)/2);
    last = left;
    for (i = left+1; i <= right; i++)
        if ((*comp)(v[i], v[left]) < 0)
            swap(v, ++last, i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp);
}
```

`int (*comp)(void *, void *)`

`(*comp)(v[i], v[left])`

`if ((*comp)(v[i], v[left]) < 0)`

`int *comp(void *, void *) /* POGREŠNO */`

```
#include <stdlib.h>

/* numcmp: compare s1 and s2 numerically */
int numcmp(char *s1, char *s2)
{
    double v1, v2;

    v1 = atof(s1);
    v2 = atof(s2);
    if (v1 < v2)
        return -1;
    else if (v1 > v2)
        return 1;
    else
        return 0;
}
```

```
void swap(void *v[], int i, int j;)
{
    void *temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```

5.12 Komplikovane deklaracije

```
int *f();    /* f: funkcija koja vraća pokazivač na int */
```

```
int (*pf)(); /* pf: pokazivač na funkciju koja vraća int */
```

```
char **argv
```

argv: pointer to pointer to char

```
int (*daytab)[13]
```

daytab: pointer to array[13] of int

```
int *daytab[13]
```

daytab: array[13] of pointer to int

```
void *comp()
```

comp: function returning pointer to void

```
void (*comp)()
```

comp: pointer to function returning void

```
char ((*x())[])()
```

x: function returning pointer to array[] of
pointer to function returning char

```
char ((*x[3])())[5]
```

x: array[3] of pointer to function returning
pointer to array[5] of char

(*pfa[])(

```
/* dcl: parse a declarator */
void dcl(void)
{
    int ns;

    for (ns = 0; gettoken() == '*'; )      /* count *'s */
        ns++;
    dirdcl();
    while (ns-- > 0)
        strcat(out, " pointer to");
}
/* dirdcl: parse a direct declarator */
void dirdcl(void)
{
    int type;

    if (tokentype == '(') {                 /* ( dcl ) */
        dcl();
        if (tokentype != ')')
            printf("error: missing )\n");
    } else if (tokentype == NAME)           /* variable name */
        strcpy(name, token);
    else
        printf("error: expected name or (dcl)\n");
    while ((type=gettoken()) == PARENS || type == BRACKETS)
        if (type == PARENS)
            strcat(out, " function returning");
        else {
            strcat(out, " array");
            strcat(out, token);
            strcat(out, " of");
        }
}
```

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

#define MAXTOKEN 100

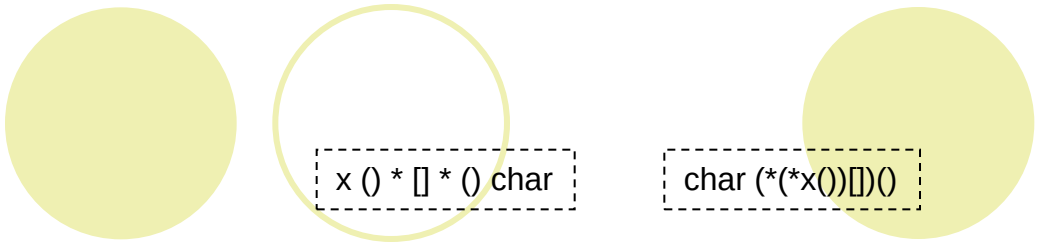
enum { NAME, PARENS, BRACKETS };

void dcl(void);
void dirdcl(void);

int gettoken(void);
int tokentype; /* type of last token */
char token[MAXTOKEN]; /* last token string */
char name[MAXTOKEN]; /* identifier name */
char datatype[MAXTOKEN]; /* data type = char, int, etc. */
char out[1000];
main() /* convert declaration to words */
{
    while (gettoken() != EOF) { /* 1st token on line */
        strcpy(datatype, token); /* is the datatype */
        out[0] = '\0';
        dcl(); /* parse rest of line */
        if (tokentype != '\n')
            printf("syntax error\n");
        printf("%s: %s %s\n", name, out, datatype);
    }
    return 0;
}
```

```
int gettoken(void) /* return next token */
{
    int c, getch(void);
    void ungetch(int);
    char *p = token;

    while ((c = getch()) == ' ' || c == '\t')
        ;
    if (c == '(') {
        if ((c = getch()) == ')') {
            strcpy(token, "()");
            return tokentype = PARENS;
        } else {
            ungetch(c);
            return tokentype = '(';
        }
    } else if (c == '[') {
        for (*p++ = c; (*p++ = getch()) != ']'; )
            ;
        *p = '\0';
        return tokentype = BRACKETS;
    } else if (isalpha(c)) {
        for (*p++ = c; isalnum(c = getch()); )
            *p++ = c;
        *p = '\0';
        ungetch(c);
        return tokentype = NAME;
    } else
        return tokentype = c;
}
```



`x () * [] * () char`

`char (*(x())[])()`

```
/* undcl: convert word descriptions to declarations */
main()
{
    int type;
    char temp[MAXTOKEN];

    while (gettoken() != EOF) {
        strcpy(out, token);
        while ((type = gettoken()) != '\n')
            if (type == PARENS || type == BRACKETS)
                strcat(out, token);
            else if (type == '*') {
                sprintf(temp, "(*%s)", out);
                strcpy(out, temp);
            } else if (type == NAME) {
                sprintf(temp, "%s %s", token, out);
                strcpy(out, temp);
            } else
                printf("invalid input at %s\n", token);
    }
    return 0;
}
```