# C programavimo kalba

**9 paskaita**

**(I/O funkcijos, komandinė eilutė, struktūros)**

# Duomenų failo pavyzdys

```
#include<stdio.h>

void main( )
{ FILE *fi;        /* aprašo rodyklę į failą*/
  int age;
     if( (fi = fopen("testas.txt", "r") ) != NULL)
     { /*  atidaro bylą skaitymui ir tikrina ar yra tokia byla  */
         fscanf ( fi, "%d", &age );
         fclose ( fi );
         fi = fopen("data.txt", "a");                 /*papildymas*/
        fprintf (fi,  "test is %d.\n", age);        /* fi nurodo į data*/
       fclose(fi);
     }
 }
```

# Duom. nuskaitymo iš failo pavyzdys

```
#include <stdio.h>

const Kiek = 15;
FILE *F;
void Sp (int *, int);
```

```
void main() {
int  A[Kiek];
int n = 0;
F = fopen("duomenys.dat", "r");
   if (F == NULL)
        printf("Failas neatidarytas\n");
   else {
        while (! feof (F)  && ( n < Kiek) )
        {  fscanf ( F, "%d", &A[n]); n++; }
        Sp(A, n);
          }
fclose (F);  }

void Sp(int *X, int n) {
int i = 0;
while ( i < n )
    printf("%d  ", X[i++]);
}
```

# I/O funkcijos

```
int getc (FILE *in_file);
```

Funkcija skirta simbolio nuskaitymui iš failo į kurį rodo rodyklė in_file. Grąžinamas einamosios pozicijos simbolis ir perstumiama pozicijos identifikatorius per vieną simbolį į priekį.

```
int fgetc (FILE *in_file);
```

Funkcija skirta simbolio nuskaitymui iš failo į kurį rodo rodyklė in_file. getc() ir fgetc() funkcijos identiškos.

```
int putc (int simbolis, FILE *out_file);
```

Funkcija skirta simbolio įrašymui į failą į kurį rodo rodyklė out_file. Funkcija grąžina įrašyto simbolio ASCII kodą. Jei įrašymas nepavyko, grąžinamas EOF.

```
int fputc (int simbolis, FILE *out_file);
```

Funkcija skirta simbolio įrašymui į failą į kurį rodo rodyklė out_file. Funkcija identiška putc().

# Duom. nuskaitymo iš failo pavyzdys

```
#include<stdio.h>

void main( )
{
FILE *in;        /* aprašo rodyklę į  failą*/
int ch;
if ( (in = fopen("test.txt", "r") ) != NULL )
/*atidaro failą skaitymui ir tikrina ar yra toks failas*/
{
     while( (ch= getc(in) ) != EOF )
          putc(ch, stdout);              /* išveda failo turinį į ekraną*/
          fclose(in);                    /* uždaro failą*/
}
else
printf("Failo test.txt atidaryti nepavyko... \n");
}
```

# Eilučių nuskaitymas/įrašymas

char ***fgets**(char *str, int n, FILE *in_file);

Funkcija nuskaito iš failo, į kurį rodo rodyklė *in_file*, ne daugiau nei *n-1* simbolį ir priskiria jį eilutei *str*. Funkcija grąžina eilutę str, o nesekmės atveju NULL.

int **fputs**(const char *str, FILE *outfile);

Funkcija įrašo į failą, į kurį rodo rodyklė *out_file* eilutę *str*. Nulinis simbolis neįrašomas. Funkcija grąžina neneigiamą reikšmę arba EOF nesekmės atveju.

```
#include<stdio.h>
#define MAXLIN 80
void main()
{
   FILE *f1;
   char string[MAXLIN];
   f1= fopen("story.txt","r");
      while( fgets(string, MAXLIN, f1) != NULL)
         puts(string);
}
```

# Pavyzdys

Funkcija **gets()** netikrina įvedamų simbolių skaičiaus, todėl galima perpildyti eilutę. Siekiant to išvengti patartina naudoti funkciją fgets(), nurodant *stdin* srautą.

```
#include <stdio.h>
char input[20];
   main()
{
     puts("Iveskite teksta:");
     gets(input);
     printf("Jusu ivestas tekstas : %s\n",input);
    return 0;
}
```

# Pavyzdys

```
#include<stdio.h>
#include<string.h>
 void main()
{
  char str[10];
  int i;
  printf("Iveskite eilute: ");
       fgets(str, sizeof(str), stdin);      // nuskaitome eilutę iš klaviatūros
  i = sizeof(str) – 1;

  if ( str[i] == '\n' )                    // pasitikrinam ar paskutinis eilutės
      str[i] = '\0';                       //simbolis '\n', jei taip, keičiam jį į '\0'
  puts(str);
}
```

# Neformatuotas įvedimas/išvedimas

int **fread**(void *buf, int t, int n, FILE *infile);

Funkcija skaito iš failo su rodykle *infile* į buferį *buf*, *n* duomenų blokų, kurių kiekvienas turi *t* baitų. Sėkmės atveju, funkcija grąžina perskaitytų baitų kiekį, nesekmės 0.

int **fwrite**(void *buf, int t, int n, FILE *outfile);

Funkcija įrašo į failą su rodykle *outfile n* blokų po *t* baitų iš buferio *buf*. Sėkmės atveju, funkcija grąžina įrašytų baitų kiekį, nesekmės 0.

int **sscanf**(const char*buffer, const char* format [, address,…]);

Funkcija veikia analogiškai scanf(), tačiau skanavimas ir formatuotas įvedimas atliekamas iš eilutės buffer. Gražinamas sėkmingai nuskaitytų laukų skaičių. Naudinga funkcija analizuojant eilutes. Verta naudoti atliekant savarankiškas užduotis.

# Pavyzdys

```
#include <stdio.h>

int main ()
{
   char sakinys []="Petrui sukako 20 metu";
   char str [20];
   int i;

   sscanf (sakinys, "%s %*s %d", str, &i);
   printf ("%s -> %d\n", str, i);
   return 0;
}
```

Ekrane: Petrui -> 20

# Pozicionavimo funkcijos

```
long int ftell(FILE *infile);
```

Funkcija grąžina einamąją pozicijos reikšmę faile.
Nesekmės atveju grąžinamas -1.

```
int fseek(FILE *infile, long offset, int from);
```

Funkcija keičia poziją faile su rodykle *infile*, perstumdama *offset* pozicijų į priekį nuo *from* pozicijos.

Argumentas *from* gali būti:

| | | |
|---|---|---|
| SEEK_SET | (=0) | t.y. failo pradžia; |
| SEEK_CUR | (=1) | einamoji pozicija faile; |
| SEEK_END | (=2) | failo pabaiga. |

Funkcija grąžina 0, jei pozicija pakeista ir ne nulį, jei įvyko klaida.

```
void rewind(FILE* infile);
```

Funkcija pastato failo pozicijos identifikatorių į failo pradžią.

# Pavyzdys

```
#include<stdio.h>

void main( )
{
   FILE *in;
   int ch;
   if ( (in = fopen("test.txt", "r") ) != NULL )
   {    ch= getc(in);
        putc(ch, stdout);
         fseek(in, 2, SEEK_CUR);
        ch= getc(in);
        putc(ch, stdout);
        fseek(in, 0, SEEK_SET);
        ch= getc(in);
         putc(ch, stdout);
         fclose(in);
   }
}
```

# Komandinė eilutė

Kartais reikia paleisti programą iš *komandinės eilutės* kartu nurodant ir programos argumentus. Norint, kad programa t.y. *main()* f-ja juos nuskaitytų, būtina apibrėžti *main()* funkcijos formalius argumentus:

```
int argc;
char *argv[ ];
char *envp[ ];
```

Pavyzdžiui:

**void main( int argc, char *argv[ ], char **envp )**

argc – argumentų skaičius komandinėje eilutėje. Programos pavadinimas taip pat įskaičiuotas.

argv – adresų masyvas, kurio elementai – tai adresai eilučių, kurios atitinka argumentus.

argv[0] – programos pavadinimas;

argv[argc -1] – paskutinis argumentas iš komandinės eilutės.

# Komandinės eilutės argumentai

envp – tai rodyklė į eilučių masyvą, apibrėžiantį programos vykdymo aplinką. Tai gali būti failai:

.cshrc . profile (UNIX OS),
autoexec.bat, config.sys, autoexec.nt ar kiti (DOS, Windows).

getenv() ir putenv() funkcijos leidžia nuskaityti ir modifikuoti aplinkos failus.

Dažniausiai naudojami *argc* ir *argv* kintamieji.

Argumentai turi būti atskiriami tarpo arba horizontalios tabuliacijos simboliu. Jei reikia perduoti argumentą, kuris turi tarpą ar tabuliaciją, tuomet toks argumentas talpinamas tarp dvigubų kabučių " ".

---

Pavyzdžiui:

*programa.exe 25 "ab c" 100*

# Komandinės eilutės argumentų interpretavimas

| Komandinė eilutė | arg[1] | argv[2] | argv[3] |
|---|---|---|---|
| "a b c" d e | a b c | d | e |
| "ab\"c" "\\" d | ab"c | \ | d |
| a\\\b d"e fg" h | a\\\b | de fg | h |
| a\\\"b c d | a\"b | c | d |
| a\\\\"b c" d e | a\\b c | d | e |

# Komandinės eilutės pavyzdys

```
#include <stdio.h>

main( int argc, char *argv[ ], char **envp )
{    int count;
     printf ( "\nKomandines eilutes argumentai:" );

     for( count = 0; count < argc; count++ )
            printf( " argv[%d]   %s\n", count, argv[count] );

      printf( "\nEnvironment variables:" );

       while ( *envp != NULL )
              printf ( "  %s\n", *(envp++) );
       return 0;
   }
```

Ekrane

```
Komandines eiltutes argumentai:  argv[0]   C:\MSC\TEST.EXE
Environment variables: COMSPEC=C:\NT\SYSTEM32\CMD.EXE
PATH=c:\nt;c:\binb;c:\binr;c:\nt\system32;c:\word;c:\help
TEMP=c:\tmp
TMP=c:\tmp
```

# Komandinės eilutės pavyzdys

```
#include <stdio.h>
FILE *input, *output;
main( int argc, char *argv[ ])
{if ( argc < 3)
            { printf("Blogai ivesta komandinė eilute!\n");
              for( int count = 0; count < argc; count++ )
                printf( "[%d] argumentas  %s\n", count+1, argv[count]);
                printf( "Truksta %d argumentu\n", 3-argc);
             return 1;}
if ( (input=fopen(argv[1], "r")) == NULL )
{     printf("Duomenu failas %s neatidarytas\n", argv[1]);
       return 1;}
if ( (output=fopen(argv[2], "w")) == NULL)
{  printf("Rezultatu failas %s neatidarytas\n", argv[2]);
   return 1;}
printf ("Viskas tvarkoje!\n");
return 0; }
```

# Struktūros

Visi kintamieji, iki šiol kuriais naudojotės, priklausė baziniams C/C++ duomenų tipams:

- Sveiko tipo (int) kintamieji ir konstantos;
- Realaus tipo (float) kintamieji ir konstantos;
- Dvigubo tikslumo (double) kintamieji ir konstantos;
- Simbolinio tipo (char) kintamieji ir konstantos;

Šie duomenys galėjo sudaryti masyvus, tačiau masyvo elementais gali būti tik to paties bazinio tipo duomenys.

**Struktūra** – tai vienodo arba skirtingo tipo kintamųjų rinkinys.

**Struktūra** priklauso išvestinių (vartotojo sukurtam) duomenų tipui.

# Struktūra realiame pasaulyje

Sąrašai

- Įmonės darbuotojai
  - Vardas, pavardė, adresas, gim.metai, išsilavinimas, paso Nr.
- Telefono numerių
  - Vardas, pavardė, adresas, tel.Nr.

Kartotekos

- Knygų, prekių, CD ...

# Struktūros

**Struktūros sintaksė:**

**struct** [*struktūrinio tipo vardas*] **{** [*laukų aprašų sąrašas*] **}**
[*kintamųjų sąrašas*];

*Laukų aprašų sąrašų* elementai atskiriami **kabliataškiais**, o *kintamųjų sąrašo elementai* – **kableliais**. Jei apraše nėra kintamųjų sąrašo, jis apibrėžia tik naujų struktūrinių duomenų tipą, tačiau jo realizacijoms atmintyje vietos neskiria.

Atskiro struktūros tipo apibrėžimo ir realizavimo pavyzdys:

```
struct telefonas { char vardas[30]; unsigned long int tel; } asmeninis;
```

# Struktūros

Struktūros elementų inicializavimas:

```
struct  Mano {
                int var1;
                float var2;
                char var3;}  pirmas = { 23, 33.4f, 'd'};
```

Kreipiantis į struktūrų elementus vartojami sudėtiniai vardai:

struktūros_vardas**.**lauko vardas

**Pavyzdys**

```
struct house
{ short RegNr;
   char gatve[50];
   char numeris[6];
   bool parking;
} namas;
```

```
Namas.RegNr = 432;
strcpy(Namas.gatve, "Uosių");
strcpy(Namas. numeris, "23");
Namas.parking = true;
```