C programavimo kalba

13 paskaita

(Funkcijos iš stdlib.h, ctype.h, pavyzdžiai)

Funkcija *system()*

int system (const char * *command* **);**

Iškviečia komandinį procesorių, kad būtų įvykdyta komanda, nurodyta argumente.

Argumentai.
command - nurodoma kaip eilutė t.y. dvigubose kabutėse.

Grąžinama reikšmė.
0 – jei komanda įvykdyta sėkmingai, 1 – jei įvyko klaida.

```
/* Sistemines komandos dir vykdymas */
#include <stdio.h>
#include <stdlib.h>
  void main ()
{ int i;
   puts (“Bandau vykdyti komanda dir");
   i = system ("dir");
     if (i != 0)  puts (“Klaida vykdant komanda dir");
     else          puts (“\nKomanda sekmingai ivykdyta");  }
```

Funkcija *realloc()*

void * realloc (void * *ptr*, size_t *size*);

Funkcija padidina arba sumažina atminties segmento dydį, kurio adresą turi rodyklė ptr. Atmintyje esantys duomenys nesikeičia.

Argumentai.

ptr - rodyklė, turinti jau išskirto atminties segmento adresą.

size – naujas atminties segmento dydis.

Grąžinama reikšmė.

Naujo atminties segmento adresas. NULLL – jei atmintis neišskirta.

Pavyzdys

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

void main ()
{
  int input,n;
  int count=0;
  int * numbers = NULL;

  do {
     printf (“Ivesk skaiciu (arba 0 jei nori baigti): ");
     scanf ("%d", &input);
     count++;
     numbers = (int*) realloc (numbers, count * sizeof(int));
     if (numbers==NULL)
       { puts (“Klaida su atmintimi"); exit (1); }
     numbers[count-1]=input;
  } while (input!=0);

  printf (“Ivesti skaiciai masyve: ");
  for (n=0;n<count;n++)
      printf ("%d ",numbers[n]);
  free (numbers);  }
```

Funkcija *qsort()*

```
void qsort( void *buf, size_t num, size_t size,
            int ( *compare)(const void *, const void *) );
```

Funkcija išrūšiuoja masyvo elementus didėjančia tvarka naudodama *Quicksort* algoritmą.

Argumentai.

buf - masyvo, kuris bus rūšiuojamas, pavadinimas.

num – elementų skaičius masyve.

size – elemento dydis baitais.

compare – funkcija, kuri grąžina:

<0 , jei pirmas argumentas mažesnis už antrą

=0 , jei pirmas argumentas lygus antrajam

>0 , jei pirmas argumentas didesnis už antrą

Funkciją *compare reikia pasirašyti pačiam !

Pavyzdys

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

int values[] = { 40, 10, 100, 90, 20, 25 };

int compare ( const void *a, const void *b)
{
  if (*(int *)a < *(int *)b)   return -1;
  if (*(int *)a == *(int *)b) return 0;
  if (*(int *)a > *(int *)b)   return 1;
}
void main ()
{ int n;
  qsort (values, 6, sizeof(int), compare);
  for (n=0; n<6; n++)
  {
    printf ("%d ",values[n]);
  }  }
```

Funkcija *exit()*

```
void  exit ( int status );
```

Sustabdo procesą (programą).

Funkcija sustabdo programos vykdymą, uždaro atidarytus failus, ištrina buferius. Baigiant darbą, išviečiama funkcija *atexit().*

Argumentai.
status - reikšmė, kuri grąžinama baigiant programą.

Status	Reikšmė	Aprašymas
EXIT_SUCCESS	0	Normali pabaiga
EXIT_FAILURE	1	Klaida uždarant programą.

Pavyzdys

```
/* exit pavyzdys */
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    FILE * pFile;
    pFile = fopen ("myfile.txt", "r");
    if (pFile == NULL)
    {
       printf (“Klaida atidarant failą  myfile.txt \n");
       exit (1);       // išeiname iš programos su klaidos kodu
     }
     else
     {    /* tam tikri operatoriai su failu */
  }
  return 0;
}
```

Funkcija *atexit()*

int atexit (void (* *function*) (void));

Nurodo funkciją, kuri turi būti vykdoma programai baigiant darbą. Funkciją, kurios pavadinimas nurodomas kaip argumentas, yra iškviečiama, kai programa sėkmingai baigia darbą.

Argumentai.
function – funkcijos pavadinimas.

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

void fnExit (void)
{  printf ("Programa baigta\n");
}
```

```
int main ()
{  atexit ();
   printf ("Pagrindine programa.\n");
  return 0; }
```

Ekrane → Pagrindine programa.
Programa baigta

Funkcija *abort()*

void abort (void) ;

Nutraukia einamąją programą (procesą) avariniu būdu.
Programa nutraukiama, atidaryti failai neuždaromi, buferiai neištrinami.
Gražinamas klaidos kodas aukštesniajam procesui.

```
/* abort() pavyzdys */
#include <stdio.h>
#include <stdlib.h>
int main ()
{   FILE * pFile;
    pFile= fopen ("myfile.txt","r");
    if (pFile == NULL)
      {  printf ("Klaida atidarant failą myfile.txt\n");
         abort();   //nutraukiama programa
       }
  /* operatoriai, vykdomi atidarius failą */
  fclose (pFile);
  return 0; }
```

Funkcija getenv()

char * getenv (const char * *varname*);

Nuskaitoma vartotojo aplinkos kintamojo reikšmė.

Grąžinama rodyklė į eilutę, kurioje patalpinta aplinkos kintamojo reikšmė. Aplinkos kintamojo pavadinimas nurodomas, kaip funkcijos argumentas.

Grąžinama reikšmė
Eilutė su aplinkos kintamojo reikšme, jei funkcija įvykdyta sėkmingai.
NULL, jei įvyko klaida.

---

int putenv (const char * *env_varname*);

Sukuria arba modifikuoja aplinkos kintamąjį.

Prideda programos aplinkos kintamąjį ir priskiria reikšmę. Jei toks kintamasis egzistuoja, modifikuojama jo reikšmė.

Argumentai.
env_varname – eilutė, kuri turi būti užrašoma tokiu formatu:
"*kintamasis = reikšmė*"

Pavyzdys

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

int main ()
{  char * buffer;
   buffer = getenv ("PATH");     // nuskaitoma kintamojo PATH reikšmė
      if ( buffer != NULL )
         printf ("Failu paieskos kelias yra: %s", buffer);

  buffer = getenv ("RESOURCES");
      if (buffer == NULL)
     {
         putenv ("RESOURCES=www.cplusplus.com");
         puts ("Aplinkos kintamasis pridetas sekmingai");
      }
     else
         puts ("Toks aplinkos kintamasis jau egzistuoja");

  return 0;
}
```

ctype.h funkcijos

ctype.h faile talpinamos funkcijos, leidžiančios nustatyti simbolio tipą.

int **isalnum**(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių:

a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

---

int **isalpha**(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių:

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

```
char c;
scanf( "%c", &c );
if( isalpha(c) )
printf( “Ivesta abeceles raide\n" );
```

ctype.h funkcijos

int iscntrl(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių:

BEL(skambutis) BS(BackSpace) CR(Carriage Return) FF (Form Feed)

HT (Horizontal Tab) NL (NewLine) VT (Vertical Tab)

---

int isdigit(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių :

0 1 2 3 4 5 6 7 8 9

---

int islower(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių :

a b c d e f g h i j k l m n o p q r s t u v w x y z

ctype.h funkcijos

int isupper(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių :

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

---

int isspace(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių :

CR FF HT NL VT space

---

int ispunct(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra vienas iš simbolių :

! " # % & ' ( ) ; < = > ? [\ ] * + , - . / : ^ _ { | } ~

ctype.h funkcijos

int tolower(int c);

Funkcija grąžina mažąją raidę, jei ji egzistuoja ir jei ji didžioji. Priešingu atveju grąžina *c*.

int toupper(int c);

Funkcija grąžina didžiąją raidę, jei ji egzistuoja ir jei ji mažoji. Priešingu atveju grąžina *c*.

int isprint(int c);

Funkcija grąžina nenulinę reikšmę, jei *c* yra *space* arba simbolis, kuris grąžina nenulinę reikšmę iš isgraph() funkcijos.

Kursas baigtas

Atsitiktinių skaičių generatorius

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
const int KIEKIS = 10;

void main()
{ time_t laikas;
  struct tm *laikas_info;
  int A[KIEKIS];
      laikas=time(NULL);
      laikas_info=localtime(&laikas);
  printf("Sekundes: %d\nLaikas %2d:%02d\n\n", laikas, laikas_info->tm_hour, laikas_info->tm_min);
       srand(laikas);
  for (int i =0; i< KIEKIS; i++)
  {     A[i] = rand();
        printf("%d ", A[i]);     }
  printf("\n");

  for (i =0; i< KIEKIS; i++)
  {     printf("%d ", A[i]%100);    }
    printf("\n");  }
```

Spausdinimas spausdintuvu LPT

```
#include <stdio.h>
#define PRINTER_IS_REMOTE YES

void main()
{
if (PRINTER_IS_REMOTE)
    {
        system("NET USE LPT1 /d");  // atlaisvinam prievada
        system("net use lpt1 \\\\Master\\hpLaser"); // prijungiam prievadą LPT1
    }

    FILE *ptr = fopen("LPT1","w");
    if (ptr == NULL)
     { printf("Klaida prijungiant spausdintuva...\n"); exit(1); }
    fprintf(ptr,"Testinis spausdinimas is C programos\n");

    fclose(ptr);
}
```

Masyvai ir funkcijos

Programa skaičiuoja vienmačio masyvo min + max bei visų elementų suma. Duomenys nuskaitomi iš failo.

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

int *ivesti(int *);          // duomenų nuskaitymas
void sp(int *, int );        // kontrolinis duomenų spausdinimas
int suma( int *, int);       // sumavimai

int main()
{    int k = 0, *A;
 A=ivesti(&k);
 sp(A, k);
 printf ("Visu elementu suma=%d\n", suma (A, k));
 return 0;
}
```

Tęsinys

```
int *ivesti(int *m)
{ FILE *F;
 int a, *C, i;
 F= fopen ("duomenys.txt", "r");
    while (!feof(F))
       {  (*m)++;
           fscanf(F,"%d",&a);
        }

C = (int *) malloc(sizeof(int)*(*m));
fseek (F, 0L, SEEK_SET);

    for ( i = 0; i<*m; i++)
     {fscanf(F,"%d", C[i]);
     }
fclose (F);
return C;
}
```

Tęsinys

```
int suma (int *D, int k)
{  int i = 0, s = 0, max, min;
    max = min = D[0];
    while ( i < k)
    {   if (max < D[i])
            max = D[i];
        if (min > D[i];
            min = D[i];
        s +=D[i++]; }
  printf ("\n Min+Max suma = %3d\n", min+max);
return s;
}
```

```
void sp (int *x, int n)
{ for (int i =0; i < n; i++)
  printf ("Elementas [%d]=%d\n", i, x[i]);
}
```

Dvimatis masyvas

Užduotis:
Duomenų faile surašytos taškų, esančių koordinačių plokštumoje, koordinatės **(x, y)**. Reikia duomenis surašyti į matricą, kur pirmame stulpelyje būtų taškų **x** koordinatės, antrame stulpelyje **y** koordinatės.

Parašyti funkciją, kuri matricos trečiame stulpelyje surašytų taškų atstumus iki koordinačių pradžios taško. Ketvirtame stulpelyje pažymėti, kokiam ketvirčiui taškas priklauso. Nuliu žymėti taškus, esančius ant ašių.

Rezultatų faile atspausdinti matricos duomenis, užrašant kiekvieno stulpelio prasmę nusakančius pavadinimus ir numeruojant eilutes. Gale parašyti, kiek kuriame ketvirtyje yra taškų, ir kiek yra taškų ant ašių.

Dvimatis masyvas

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

typedef float mas[][4];       // sukuriamas naujas kintamojo tipas

const char *Duomenys = "Duom8.dat";
const char *Rezultatai = "Duom8.rez";

void RezFailas();                    // Rezultatų failo paruošimas
int Failas();                        // Randa kiek yra duomenų
mas *Ivesti(int);                    // Duomenų įvedimas
void Spausdinti(mas *, int, int); // Rezultatų spausdinimas
void Atstumai(mas *, int);           // Taškų atstumai
void Vieta(mas *, int);              // Taškų padėtis plokštumoje
```

Dvimatis masyvas

```
void main() {
mas *A;          //rodyklė, kuriai bus priskirtas dvimačio masyvo adresas
int n;
RezFailas();
if( (n = Failas()) == 0)
{  printf (" Duomenu nerasta\n“);
  exit(0); }
A = Ivesti(n);
   if (A == NULL) {
      printf(“Truksta atminties \n";)
      exit(1); }
printf( "Masyvas ivestas n = “, n);
Spausdinti(A, n, 0);
Atstumai(A, n);
Spausdinti(A, n, 1);
Vieta(A, n);
Spausdinti(A, n, 2);
}
```

Dvimatis masyvas

```
// Paruošia rezultatų failą.
void RezFailas() {
FILE *F;
if(F = fopen(Rezultatai, "w")) {
fprintf(F, "\tREZULTATU  FAILAS\n");
fclose(F); }
}
```

```
// Patikrina, ar yra duomenų failas ir kiek jame taškų
int Failas() {
FILE *F;
int n = 0, k;
if ((F = fopen(Duomenys, "r")) == NULL ) {
printf( "Duomenu failo nera\n";)
return 0; }
while(!feof(F)) {
n++;
fscanf(F, "%d%d\n", &k, &k); }
fclose(F);
return n;  }
```

Dvimatis masyvas

```
// Skaito duomenis iš failo į masyvą
mas *Ivesti(int n) {
FILE *F;
mas *A;
if ((F = fopen(Duomenys, "r")) == NULL) {
    printf( "Duomenu failas neatidarytas\n");
    return NULL; }
A = (mas *) malloc(n*4*sizeof(float));
if(A == NULL) {
    printf( "Truksta masyvui atminties\n");
    return NULL; }
n = 0;
while(!feof(F)) {
    fscanf(F, "%f%f\n", &(*A)[n][0], &(*A)[n][1]);
    n++; }
fclose(F);
return A;
}
```

Dvimatis masyvas

// **Skaičiuojami taškų atstumai iki koordinačių pradžios taško**

```
void Atstumai(mas *D, int n) {
float x, y;
for (int i = 0; i<n; i++) {
   x = (*D)[i][0];
   y = (*D)[i][1];
   (*D)[i][2] = sqrt(x*x + y*y); }
}
```

Dvimatis masyvas

// **Nustatoma taško padėtis koordinačių plokštumoje**
/* 1–4 nurodo ketvirčio numerį, o 0 – kad taškas yra ant vienos iš ašių. */

```
void Vieta(mas *D, int n) {
float x, y;
for (int i=0; i<n; i++) {
x = (*D)[i][0];
y = (*D)[i][1];
if ((x>0) && (y>0)) (*D)[i][3] = 1;
else if ((x>0) && (y<0)) (*D)[i][3] = 4;
else if ((x<0) && (y>0)) (*D)[i][3] = 2;
else if ((x<0) && (y<0)) (*D)[i][3] = 3;
else (*D)[i][3] = 0; }
```

Dvimatis masyvas

```
void Spausdinti(mas *C, int n, int k) {
FILE *F;
int i;
if ((F = fopen(Rezultatai, "a")) == NULL) {
cout << "Rezultatu failas neatidarytas\n";
getch(); return; }
switch(k) {
case 0:
fprintf(F, "Duomenys\n");
fprintf(F, "**********\n");
fprintf(F, " Nr. x y \n");
break;
case 1:
fprintf(F, "\nDuomenys ir atstumai iki koord.
pradzios\n");
fprintf(F, "**********\n");
fprintf(F, " Nr. x y Atstumas \n");
break;
```

Dvimatis masyvas

```
case 2:
fprintf(F, "\nDuomenys, atstumai ir tasko vieta
koord. plokstumoje\n");
fprintf(F, "**********************************\n");
fprintf(F, " Nr. x y Atstumas Vieta \n");
break;
}
for(i=0; i<n; i++) {
switch(k){
case 0: fprintf(F," %3i %7.2f %7.2f \n", i+1,
(*C)[i][0],(*C)[i][1]);
break;
case 1: fprintf(F, "%3i %7.2f %7.2f %7.2f \n",
i+1, (*C)[i][0],(*C)[i][1],(*C)[i][2]);
break;
case 2:
fprintf(F, "%3i %7.2f %7.2f %7.2f %7.2f \n",
i+1, (*C)[i][0],(*C)[i][1],(*C)[i][2],(*C)[i][3]);
break;}}
```

Dvimatis masyvas

```
switch(k) {
41
case 0:
fprintf(F, "**********************\n");
break;
case 1:
fprintf(F,
"******************************\n");
break;
case 2: fprintf(F,
"**********************************
****\n");
break;
}
fclose(F);
}
```