

Programmēšanas valodas

Programming languages

Языки программирования

Definīcijas Programmēšanas valoda ir

- ↻ Formālā valoda, kas kalpo datora programmu aprakstam (Formālā valoda ir simbolu secību kopa, ko apraksta ar precīzi formulētu sintakses likumu kopu...)
- ↻ Zīmju sistēma, kas nodrošina lietotāja saskarsmi ar datoru dažādu problēmu risināšanā

2

Definīcijas Programmēšanas valoda ir

- ↻ Simbolu secība kādam alfabētam, kas apmierina sintakses likumus un uzdod izskaitļojumu secību ar semantikas likumiem
- ↻ Algoritmiska valoda uzdevumu risināšanas procesu formalizētam pierakstam programmu veidā

3

Secinājumi no definīcijām

1. Programmēšanas valodas ir vajadzīgas, lai risinātu dažādas problēmas ar datoru
2. Programmēšanas valodas lieto programmu (un algoritmu) pierakstam
3. Programmēšanas valoda ir formāla valoda, kas ietver:
 - simbolu secību (alfabēts);
 - likumu kopu (uzdod sintaksi un semantiku)

4

Kāpēc pastāv vairākas valodas versijas ?



- ↻ Vēlme uzlabot valodu
- ↻ Nepieciešamība izlabot kļūdas
- ↻ Vēlme pielāgot valodu noteikta uzdevuma tipa risināšanai
- ↻ Nespēja realizēt valodu pilnībā noteikta tipa datoram

5

Programmēšanas valodu raksturīgas īpašības

- | | |
|---|----------------|
| ↻ Jauda | ↻ Pilnība |
| ↻ Līmenis | ↻ Elastīgums |
| ↻ Konceptuālais veselums
(jēdzienu taupība,
ortogonalitāte,
vienveidība) | ↻ Vienkāršums |
| ↻ Drošums | ↻ Pārnesamība |
| ↻ Lasamība | ↻ Efektivitāte |

6

Programmēšanas valodu pamatjēdzieni

Alfabēts – rakstzīmju kopa, kas parasti ietver burtus, ciparus un speciālās zīmes

Sintakse – noteikumi, kas nosaka atļauto konstrukciju izveidošanu, kā arī rakstzīmju izvietošanu secībā programmā

Semantika – noteikumi, kas nosaka kādas operācijas un kādā secībā datoram jāizpilda, strādājot pēc programmas

7

Programmēšanas valodu semantika

Program-
mēšanas
valoda

Datu apraksta
līdzekļi

Darbību
apraksta
līdzekļi

Datu apraksta
pamatlīdzekļi

Datu abstraktie tipi

Papildus iespējas

Darbību apraksta
pamatlīdzekļi

8

Programmēšanas valodas Datu apraksta pamatlīdzekļi

Datu
apraksta
pamat-
līdzekļi

Vien-
kāršie
datu
tipi

Uzskai-
tāmie

Loģisks
Simbolu
Lietotāja

Skait-
liskie

Vesels
Reāls

Strukturētie
datu tipi

Masīvi, Ieraksti,
Apvienības

9

Programmēšanas valodas Darbību apraksta pamatlīdzekļi

Darbību
apraksta
pamat-
līdzekļi

Izteiksmes

Piešķiršanas operatori

Vadības operatori

Bloki

Apakšprogrammas

Pakotnes

Pārejas operators

Nosacījuma operatori

Cikla operatori

Procedūras

Funkcijas

10

Programmēšanas valodas Papildus iespējas

Papildus
iespējas

Datņu (failu) apstrāde
Ārkārtējo stāvokļu apstrāde
Paralēlā apstrāde
Makrokomandu apstrāde

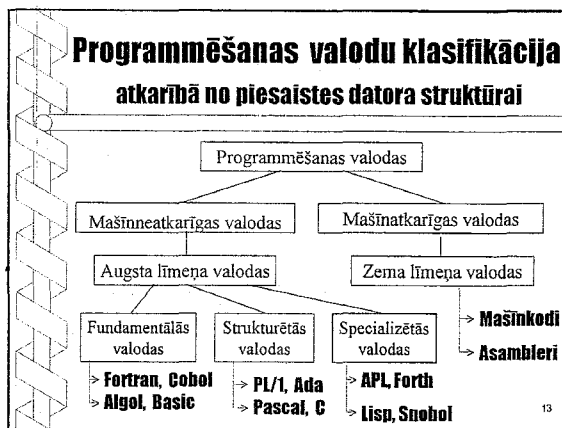
11

Programmēšanas valodu klasifikācija

- ↻ Piesaiste datora struktūrai
- ↻ Lietojumsfēra
- ↻ Funkcionēšanas vide
- ↻ Valodas uzbūve

12

Programmēšanas valodu klasifikācija atkarībā no piesaistes datora struktūrai



Mašīnneatkarīgas valodas

Mašīnneatkarīga valoda ir programmēšanas valoda, kuras struktūra un izmantojamie līdzekļi nav saistīti ar vienu konkrētu datoru

Augsta līmeņa valoda ir datorneatkarīga programmēšanas valoda, kas ir tuva dabīgai valodai

Fundamentālās valodas: lielas programmu bibliotēkas, plašs pielietojums

Strukturētās valodas: strukturētās loģiskas konstrukcijas, stingras procedurālās un datu struktūru prasības

Specializētās valodas: neparastas sintakses formas, ierobežots pielietojums

Mašīnatkarīgas valodas

Mašīnatkarīga valoda ir programmēšanas valoda, kas atkarīga no konkrēta datora īpatnībām

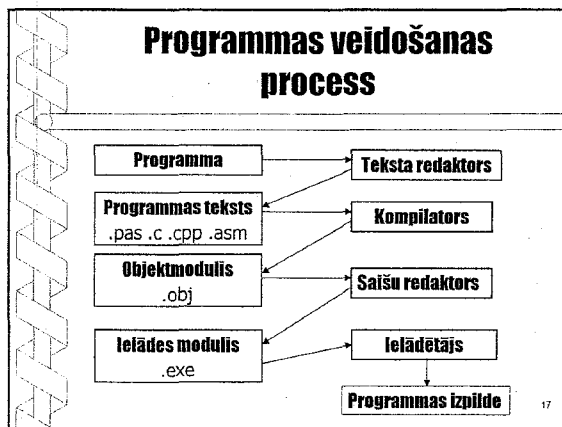
Mašīnvaloda (Mašīnkods) ir programmēšanas valoda, kuras komandas tieši sakrīt ar datora izpildāmajām operācijām

Asamblervaloda ir programmēšanas valoda, kuras komandu struktūru nosaka mašīnvalodas komandu un datu formāti, kā arī datora arhitektūra

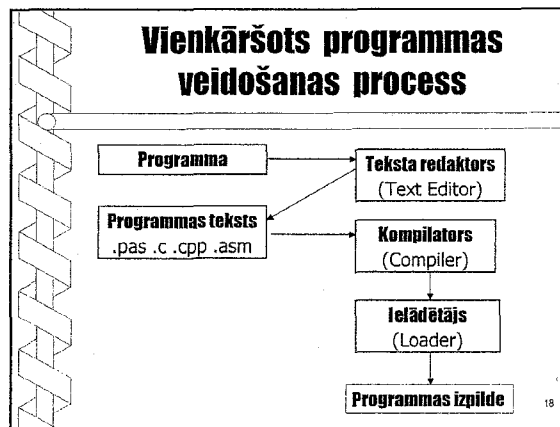
Programmēšanas valodas atkarībā no valodas uzbūves

- Procedurālās valodas
- Uz datiem orientētās valodas
- Objektorientētās valodas
- Neprocedurālās valodas

Programmas veidošanas process



Vienkāršots programmas veidošanas process



- ```

1. if (gads % 4 == 0 && gads % 100 != 0 !!
 gads % 400 == 0)
 printf (" %d - garais gads.\n", gads);

2. if (x <= 0)
 { y = x;
 z = -x; }
else
 { y = 1 / x;
 z = sort (x); }

3. if (a == b)
 if (b == c) k = 1;
 else k = 2;
else
 if (a == c) k = 3;
 else k = 4;

```

### Programmēšanas valodas C cikla operators while

**while (izteiksme) operators;**

```
include <stdio.h>
main()
{ float grad, rad;
 grad = 0;
 while (grad <= 90)
 { rad = (3.14 * grad) / 180;
 printf ("%2.0f %f\n", grad, rad);
 grad +=5; /*grad=grad+5;*/
 }
}
```

25

### Programmēšanas valodas C cikla operators do-while

**do operators while (izteiksme);**

```
include <stdio.h>
main()
{ float grad, rad;
 grad = 0;
 do { rad = (3.14 * grad) / 180;
 printf ("%2.0f %f\n", grad, rad);
 grad +=5; }
 while (grad <= 90)
}
```

26

### Programmēšanas valodas C cikla operators for

**for (inicializācija; izteiksme; korekcija) operators**

```
include <stdio.h>
main()
{ float grad, rad;
 for (grad = 0; grad <= 90; grad +=5)
 { rad = (3.14 * grad) / 180;
 printf ("%2.0f %f\n", grad, rad);
 }
}
```

27

### Programmēšanas valodas C cikla operatoru piemēri

- for ( k = 1; k <= 10000; k++ );
- n = 2; for ( k = 5; n < 100 ) n = n \* k;
- for ( x=-1.5; x<=1; x+=.5)
   
for ( y=1; y<=2; y+=.2)
   
{ z = x + y;
   
printf ("%4.1f %4.1f %4.1f\n", x, y, z); }
- for ( x=-1.5, y=1; x<=1, y<=2; x+=.5, y+=.2)
   
{ z = x + y;
   
printf ("%4.1f %4.1f %4.1f\n", x, y, z);
   
}

28

### Programmēšanas valodas C cikla operatoru piemēri (turpinājums)

- # include <stdio.h>
   
# define END '#'
   
main ( )
   
{ char simb;
   
 while (( simb = getchar( )) != END)
   
 putchar ( simb );
   
}
- simb\_skaits = 0;
   
while (( simb = getchar( )) != END)
   
{ if ( simb == '' ) continue; /\* break; \*/
   
 putchar ( simb );
   
 simb\_skaits++; }

29

### Datu tipi programmēšanas valodā C

|          |           |                       |     |
|----------|-----------|-----------------------|-----|
| int      | vesels    | -32768 - 32767        | (2) |
| short    | īss       | -32768 - 32767        | (2) |
| long     | garš      | -2147483648 - 2...647 | (4) |
| float    | peldošs   | -3.4e-38 - 3.4e+38    | (4) |
| double   | dubults   | -1.7e-308 - 1.7e308   | (8) |
| char     | simbols   | -128 - 127            | (1) |
| unsigned | bez zīmes | 0 - 65535             | (2) |

30

## Skaitliskas konstantes programmēšanas valodā C

### Vesela tipa konstantes

|                   |                  |                |
|-------------------|------------------|----------------|
| • decimālās       | -15 443 +2840    | int            |
|                   | 22l -958L +10L   | long           |
|                   | 18u 12345U       | unsigned       |
| • astotnieku      | 012 0111 076L    | (10, 73, 62)   |
| • sešpadsmitnieku | 0x20 0xA5 0x1B9L | (32, 165, 441) |

### Peldoša punkta konstantes

|                               |                   |        |
|-------------------------------|-------------------|--------|
| • decimāldaļskaitļi           | -25.4 .5 +10. 0.8 | double |
| • skaitļi ar mantisu un kārtu | 142E5 2.1e7 .1E-4 |        |

31

## Simbolu konstantes programmēšanas valodā C

### Simbolu konstantes 'A' '7' '>' char

|               |        |                     |
|---------------|--------|---------------------|
| Vadības kodi: | '\n'   | - uz jaunu rindu    |
|               | '\t'   | - tabulācija        |
|               | '\a'   | - skaņas signāls    |
|               | '\f'   | - uz jaunu lappusi  |
| Citi piemēri: | '\\'   | - otrādā slīpsvītra |
|               | '\"'   | - apostrofs         |
|               | '\''   | - pēdīgas           |
|               | '\0'   | - nulles simbols    |
|               | '\x1B' | - Escape (Esc)      |

32

## Simbolu virknes (konstantes) programmēšanas valodā C

|                                                 |      |
|-------------------------------------------------|------|
| "string of characters"                          | 20+1 |
| "simbolu virkne"                                | 14+1 |
| "rindiņa"                                       | 7+1  |
| "0"                                             | 1+1  |
| ""                                              | 0+1  |
| "Kinoteātris \"Daile\" (Kinoteātris \"Daile\")" | 19+1 |
| "A"                                             |      |
| 'A'                                             |      |

33

## Mainīgo apraksts programmēšanas valodā C

**Mainīgo apraksts:** int k1, k\_first, k\_last, k\_step;  
float x, y, z;  
unsigned atzime;  
double sum, vid\_atzime;

**Apraksts ar sākumvērtībām:** int i = 0;  
float pi = 3.14, eps = 1.0e-5;  
float p2 = 2 \* pi;  
char virknes\_beigas = '\0';  
double vid\_atzime, sum = 0.0;  
const double e = 2.718281828;

## Operācijas programmēšanas valodā C

### Aritmētiskās operācijas:

- (1) - (vienvietīgais mīnuss jeb zīmes maiņa)
- (2) \*, /, % (atlikums no veselās dalīšanas)
- (3) +, -

### Palielināšanas un samazināšanas operācijas:

++, -- (palielināšana un samazināšana par 1)

### Salīdzināšanas operācijas:

<, <=, >=, >, ==, !=

### Loģiskās operācijas:

! (negācija), && (un), || (vai)

### Operācijas ar kodiem:

~ (pretējais kods), & (un), | (vai), ^, <<, >> (koda nobīde)

35

## Aritmētiskās operācijas programmēšanas valodā C

**Piemēri:** -a / b + (2.5 \* c - d) / 1.8  
4 \* k % m /\*k=5; m=2; rez=4\*/  
sqrt ( b\*b - 4\*a\*c)

### Tipu pārveidošanas shēma:

|        |               |                     |
|--------|---------------|---------------------|
| double | ← float       | int k, m, n;        |
| long   |               | sqrt ( (double) n ) |
| ↑      |               | 4 * m % (int) a     |
| ↑      |               |                     |
| ↑      |               |                     |
| int    | ← char, short |                     |

36

## Palielināšanas un samazināšanas operācijas

++ palielināšana par 1

-- samazināšana par 1

**Vienīgais operands - mainīgais**

• prefiksa forma: ++k --n

• postfiksa forma: k++ n--

**Piemēri:** k=4; x=++k; /\* k=5; x=5; \*/  
k=4; y=k++; /\* y=4; k=5; \*/  
k=4; x1=--k; /\* k=3; x1=3; \*/  
k=4; y1=k--; /\* y1=4; k=3; \*/

## Salīdzināšanas operācijas programmēšanas valodā C

**Salīdzināšanas operācijas:** <, <=, >=, >

= (vienāds), != (nav vienāds)

**Piemēri:** 10 > 2 a <= b n == 0 'a' > 'b'

**Operācijas rezultāta vērtība:** 1 ("paties") vai 0 ("nepaties")

```
#include <stdio.h>
main ()
{ int true, false;
 true = (10 > 2); false = (10 == 2);
 printf ("true = %d; false = %d; \n", true, false);
}
```

## Loģiskās operācijas programmēšanas valodā C

**Loģiskās operācijas:** ! (negācija jeb operācija "ne")  
&& (konjunkcija jeb operācija "un")  
!! (disjunkcija jeb operācija "vai")

**Operācijas rezultāta vērtība**

| A | B | !A | A && B | A    B |
|---|---|----|--------|--------|
| 0 | 0 | 1  | 0      | 0      |
| 0 | 1 | 1  | 0      | 1      |
| 1 | 0 | 0  | 0      | 1      |
| 1 | 1 | 0  | 1      | 1      |

**Piemēri:**

x < 2.5 && x >= -4.2

n <= -1 || n >= 1

10 && 2 (1)

10 && 0 (0)

## Nosacījuma operācija programmēšanas valodā C

izteiksme1 ? izteiksme2 : izteiksme3

**Piemēri:**

z = (x > y) ? x : y; /\* z = max (x, y); \*/

double a, b, c, p, s, tr\_laukums;

p = (a + b + c) / 2.0;

s = p \* (p - a) \* (p - b) \* (p - c);

tr\_laukums = (s > 0) ? sqrt(s) : 0;

## Operāciju prioritāte programmēšanas valodā C

1) ! ~ ++ -- - (vienvietīgais) (tips)

2) \* / %

3) + -

4) << >>

5) < <= >= >

6) = !=

7) & 11) !!

8) ^ 12) ? :

9) | 13) piešķires operācijas

10) && 14) ,

## Piešķires operācijas programmēšanas valodā C

mainīgais = izteiksme

**Piemēri:** int x, y, z, a=3, k=0;

x=15; y=++x%3+1; k=k+1;

z=2\*y+3\*(x=5+4\*a);

/\* x=5+4\*a; z=2\*y+3\*x; \*/

x\*=2; /\* x=x\*2; \*/

k+=1; /\* k=k+1; \*/

k/=3; /\* k=k/3; \*/

z=4\*y+(x-=12+y);

## Masīvi programmēšanas valodā C

```
int a[10]; char virkne[] = "Tā ir virkne";
double x [5], y [21]; float c [3][4];

#include <stdio.h>
int dienas[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
main ()
{ int num, i, d = 0;
 printf ("Ievadiet mēneša numuru: ");
 scanf ("%d", &num);
 for (i = 0; i < num-1; i++)
 d += dienas [i];
 printf ("%d dienas līdz %d mēnesim\n", d, num); }
```

## Rādītāji programmēšanas valodā C

**Rādītājs** ir mainīgais, kura vērtība ir adrese datora atmiņā

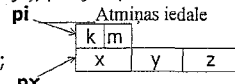
**Apraksta operatori:**

```
int *pi;
float *p1, *p2;
```

**Operācijas:** & (adreses noteikšana), ++, --, +=, -=, \* (netieša adresācija), piešķires, salīdzināšana

**Piemēri:**

```
int k, m, *pi;
float x, y, z = -1.5, *px, *p;
pi = &k; px = &x;
x = z; k = -12; y = *px; /* y=x=-1.5*/
m = *pi; /*m=-12*/ *pi = 0; /* k=0 */
p = px; *p++ = 2.5; /*x=2.5*/ (*px)++; /*x=3.5*/
```



## Masīvi un rādītāji programmēšanas valodā C

```
int a[10], *p, b; a[i] ==> *(a + i)
p = &a[0]; /* p=a; */ *p = 1; b = *(p + 1);

#include <stdio.h>
int dienas[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
main ()
{ int num, i, d = 0;
 printf ("Ievadiet mēneša numuru: ");
 scanf ("%d", &num);
 for (i = 0; i < num-1; i++)
 d += *(dienas + i);
 printf ("%d dienas līdz %d mēnesim\n", d, num); }
```

## Masīvu un rādītāju lietošanas piemēri valodā C

```
int a[100], i, sum=0; int a[100], i, sum=0;
for (i = 0; i < 100; ++i) for (i = 0; i < 100; ++i)
 scanf ("%d", &a[i]); scanf ("%d", &a[i]);
for (i = 0; i < 100; ++i) for (i = 0; i < 100; ++i)
 sum += a[i]; sum += *(a + i);
printf ("sum=%d\n", sum); printf ("sum=%d\n", sum);

float c[4][10], i, j, s=0;
for (i = 0; i < 4; ++i)
 for (j = 0; j < 10; ++j)
 s += c[i][j];
printf ("s = %f\n", s);
```

## Masīvu un rādītāju lietošanas piemēri (turpinājums)



```
#include <stdio.h>
main ()
{
 int i;
 float c[4][10], *pc, s = 0;
 printf ("Ievadiet 40 masiva elementus\n");
 for (i = 0; i < 40; ++i)
 scanf ("%f", c + i);
 for (i = 0, pc = c; i < 40; ++i, pc++)
 s += *pc;
 printf ("summa = %f\n", s);
}
```

## Funkcijas programmēšanas valodā C

**Funkcijas struktūra:**

```
< tips > <funkcijas vārds> (<parametru saraksts>)
{ ...
 return <izteiksme>; ...
}
```

**Piemērs:**

```
void autors (void)
{
 printf (" \nProgrammas autors - Juris Osis,
 1. kurss DK2 grupa\n");
}
```

**Funkcijas izsaukums:**

```
<funkcijas vārds> (<argumentu saraksts>)
```



## Funkciju piemēri programmēšanas valodā C

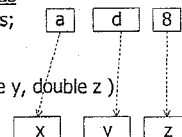
```
#include <stdio.h>
double max3 (double, double, double);
void autors (void); /* funkciju prototipi */
main ()
{ int a = 12; double d = -2.4e3, s;
 autors (); /* funkcijas autors izsaukums */
 s = max3 (a, d, 8); /* funkcijas max3 izsaukums */
 printf ("%d %f %f\n", a, d, s);
}
double max3 (double x, double y, double z)
{ double r;
 r = (x > y) ? x : y; /* if (x>y) r=x; else r=y; */
 return (r > z) ? r : z;
}
```

## Funkciju piemēri (turpinājums) programmēšanas valodā C

Funkcijas izsaukums pēc vērtības

```
int a = 12; double d = -2.4e3, s;
s = max3 (a, d, 8);
```

```
double max3 (double x, double y, double z)
{ ... }
```



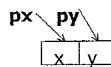
Funkcijas izsaukums pēc adreses

```
double max3r (double*, double*, double*);
s = max3r (&a, &d, &8);
```

```
double max3r (double *x, double *y, double *z)
{ ... }
```

## Funkciju piemēri (turpinājums) programmēšanas valodā C

```
void maina (int *px, int *py)
{ int k;
 k = *px;
 *px = *py;
 *py = k;
}
void maina1 (int a, int b)
{ int k;
 k = a; a = b;
 b = k; }
main ()
{ int x = 5, y = 10;
 printf ("x=%d y=%d\n", x, y);
 maina (&x, &y);
 printf ("x=%d y=%d\n", x, y);
}
maina1 (x, y);
```



Rezultāts: x=5 y=10 x=10 y=5

## Funkciju piemēri (turpinājums) programmēšanas valodā C

```
float Tr_P&S (int a, int b, int c, int *pp)
{ float p2;
 *pp = a + b + c;
 p2 = *pp / 2.;
 return (sqrt (p2*(p2-a)*(p2-b)*(p2-c)));
}
```

```
main ()
{ int a, b, c, p;
 float s, Tr_P&S (int, int, int, int*);
 scanf ("%d%d%d", &a, &b, &c);
 s = Tr_P&S (a, b, c, &p);
 printf ("\n%d %d %d\n p=%d s=%f\n", a,b,c, p,s);
}
```

## Funkciju un masīvu piemērs programmēšanas valodā C

```
include <stdio.h>
define EL_SKAITS 100
void ievade (int *, int);
int sum (int *, int);
main ()
{ int x[EL_SKAITS], skaits = EL_SKAITS;
 ievade (x, skaits);
 printf ("sum=%d\n", sum (x, skaits)); }
void ievade (int *a, int n)
{ int i; for (i = 0; i < n; ++i) scanf ("%d", &a[i]); }
int sum (int *a, int n)
{ int i, s = 0;
 for (i = 0; i < n; ++i) s += *(a + i); return s; }
```

## Rekursīvās funkcijas programmēšanas valodā C

Rekursīvā funkcija ir funkcija, kurā paredzēta vēršanās pašai pie sevis

```
main ()
{ char simbols;
 printf ("Ievadiet simbolu vai #\n");
 simbols = getchar ();
 printf ("Ievadīts %c\n", simbols);
 if (simbols != '#') main ();
 printf ("%c", simbols);
}
```

```
F 1.izs.
I 2.izs.
L 3.izs.
E 4.izs.
#
FILE #
#ELIF
```

## Atmiņas klases programmēšanas valodā C

- ☞ **auto** ( automatic - automātisks )  
Atmiņa tiek iedalīta automātiski, kad funkcija sāk darbu, un tiek atbrīvota, funkcijai darbu beidzot.
- ☞ **extern** ( external - ārējs )  
Atmiņa tiek iedalīta, kad programma sāk darbu, un tiek saglabāta līdz programmas nobeidumam.
- ☞ **static** ( statisks )  
Atmiņa tiek iedalīta, kad tiek sasniegts objekta apraksts, un tiek saglabāta līdz programmas nobeidumam
- ☞ **register** ( reģistrs )  
Objekts jāglabā kādā no procesora reģistriem nevis<sup>65</sup> datora atmiņā

## Aprakstu darbības sfēras programmēšanas valodā C

| Atmiņas klase         | Darbības sfēra           | Eksistēšanas laiks |
|-----------------------|--------------------------|--------------------|
| <b>auto</b>           | lokāls                   | Īslaicīgs          |
| <b>extern</b>         | globāls<br>(visi faili)  | pastāvīgs          |
| <b>static</b> iekšējs | lokāls                   | pastāvīgs          |
| <b>static</b> ārējs   | globāls<br>(viens fails) | pastāvīgs          |
| <b>reister</b>        | lokāls                   | Īslaicīgs          |

56

## Aprakstu darbības sfēras piemēri programmēšanas valodā C

|                                                                                                                                                                                                            |                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <p><u>Fails prog.c</u></p> <pre>#include&lt;stdio.h&gt; int i, k = 0, sum; main() { float x, y;   ....   float x;   int fun1 (y)   { static int skaits = 0; int i; extern int k;     ++skaits;   } }</pre> | <p><u>Fails fun5.c</u></p> <pre>int fun5 (a, b) { extern int k;   .... }</pre> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|

57

## Simbolu virkņu apraksts programmēšanas valodā C

```
include<stdio.h>
char text [] = "Rezultātu tabula";
main()
{ static char text1 [] = "Ievadiet paroli:";
 char string [20];
 static char letter [] = {'a', 'e', 'i', 'o', 'u', '\0' };

 printf ("Programma sāk darbu");
}
```

58

## Simbolu virkņu ievade-izvade programmēšanas valodā C

- ☞ Funkcijas **scanf** un **printf** ar formātu **s**  
char uzvards [15], \*p;  
scanf ("%s", uzvards); /\* ievadīts Juris Osis \*/  
printf ("%s", uzvards); /\* izvada Juris \*/  
printf ("%s", text); /\* Rezultātu tabula \*/
- ☞ Funkcijas **gets** un **puts (<rādītājs>)**  
p = gets (uzvards); /\* ievadīts Juris Osis \*/  
printf ("%s", uzvards); /\* izvada Juris Osis \*/  
puts (uzvards); /\* izvada Juris Osis \*/  
puts (&uzvards[6]); /\* izvada Osis \*/  
puts ( p+6); /\* izvada Osis \*/  
p = "Tā ir virkne"; puts(p+3);

59

## Darbs ar simbolu virknēm programmēšanas valodā C

|                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre># include&lt;stdio.h&gt; main() { char text [ 80 ];   int i, skaits = 0;   printf ("Ievadiet tekstu\n");   gets ( text );   for (i=0; text [i] != '\0'; ++i)   if (text [i] == ' ') ++skaits;   puts ( text );   printf ("Skaits = %d\n", skaits); }</pre> | <pre>char ch, *cptr; ch = getchar ( ); switch ( ch ) { case 'Y':   case 'y': cptr = "yes";     break;   case 'N':   case 'n': cptr = "no";     break;   default: cptr = "error"; }; puts ( cptr );</pre> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

60

## Struktūras programmēšanas valodā C

```
struct student /* strukturētā tipa apraksts */
{ char uzvards [15];
 int atzime [5];
 float vid_atzime; };
main ()
{ struct student stud; /* struktūras stud apraksts */
 int i, sum=0;
 scanf ("%15s", stud.uzvards);
 for (i=0; i<5; ++i)
 { scanf ("%d", &stud.atzime[i]);
 sum += stud.atzime[i]; }
 stud.vid_atzime = (float) sum / 5;
 printf ("%17s %f\n", stud.uzvards, stud.vid_atzime); }
```

## Atmiņas iedalīšana struktūrām programmēšanas valodā C

```
struct student stud;
uzvards atzime[0] atzime[1] atzime[4] vid_atzime
0 1 15 16 19 20 23 24 31 32 35 36 39

struct date
{ int day;
 int month;
 int year;
} d;
sizeof (struct student) = 39
sizeof (d) = 12
```

## Struktūru lietošanas piemēri programmēšanas valodā C

```
struct date { int day, month, year; } d = {13, 11, 2001};
struct student
{ char uzvards [15];
 struct date dzim_datums;
 int atzime [5];
 float vid_atzime; };
main ()
{ struct student stud, *s;
 s = &stud;
 s->atzime[0] = 9; /* stud.atzime[0] = 9; */
 s->uzvards = "Juris Osis";
 d.day = 18; }
```

## Struktūru masīvi programmēšanas valodā C

```
struct student { char uzvards [15]; int atzime [5];
float vid_atzime; };
main ()
{ struct student st [25], *s;
 int i, k, sum=0;
 float grupas_vid = 0.0;
 /* struktūru masīva ievade */
 for (k=0, s=&st[0]; k<25; ++k, ++s)
 { vid = 0; for (i=0; i<5; ++i)
 sum += s->atzime[i];
 grupas_vid += sum; st[k].vid_atzime = (float) sum / 5;
 printf ("%15s %f\n", s->uzvards, s->vid_atzime); }
 grupas_vid /= 25;
 printf ("Grupas vidējā atzime - %f\n", grupas_vid); }
```

## Struktūras un funkcijas programmēšanas valodā C

```
#include <stdio.h>
struct date { int day, month, year; };
main ()
{ struct date d; printf ("Ievadiet datumu: ");
 scanf ("%d%d%d", &d.day, &d.month, &d.year);
 printf ("%d.%d.%d - %d diena gadā\n",
 d.day, d.month, d.year, daynumb (&d)); }
int diasnas[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
int daynumb (struct date *p)
{ int i, dat, k2; dat = p->day;
 for (i=1; i< p->month; i++) dat += diasnas[i];
 if (p->month > 2) { k2 = p->year % 4 ==0 &&
 p->year % 100 != 0 !! p->year % 400 == 0;
 dat += k2; } ; return (dat); }
```

## Galvenās funkcijas parametri programmēšanas valodā C

```
include <stdio.h>
main(int argc, char*argv[])
/* argc - argumentu skaits palaišanas komandā
 argv - rādītāju masīvs, satur argumentu adreses */
{ int i;
 printf ("argc = %d\n", argc);
 for (i=0; i < argc; ++i)
 printf ("argv[%d] = %s\n", argv[i]); }

Ievadīts: Izvadīts: argc = 3
prog.exe valoda C argv[0] = prog.exe
 argv[1] = valoda
 argv[2] = C
```

## Faili programmēšanas valodā C

Faili (datne) ir datu kopa, ko glabāšanas, pārsūtīšanas vai apstrādes procesā uzskata par vienotu veselumu un kas parasti sastāv no vienkāršas struktūras ierakstiem.

Standarta faili: **stdin** - ievade no tastatūras  
**stdout** - izvade uz ekrāna  
**stderr** - ziņojumu fails par kļūdām

Faila apraksts: **FILE** <faila rādītājs>;

Faila rādītājs - norāde uz ierakstu, kas satur informāciju par failu

Piemērs: **FILE \*in, \*out;**

Programmā jāiekļauj #include <stdio.h>

67

## Failu atvēršana un aizvēršana programmēšanas valodā C

Funkcijas: **fopen** - atvērt failu

**fclose** - aizvērt failu

Failu atvēršana:

<faila rādītājs> = **fopen**(<faila nosaukums>, <režims>);

Režimi: **"r"** - fails tiks lasīts (jābūt iepriekš izveidotam);

**"w"** - notiks ierakstīšana failā (fails tiks radīts. Ja fails eksistē, dati zudīs)

**"a"** - notiks ierakstīšana faila beigās (Ja faila nav, tas tiks izveidots)

Failu aizvēršana:

**fclose** (<faila rādītājs> );

**fcloseall** ();

68

## Failu atvēršanas un aizvēršanas piemēri

```
FILE *in, *in1, *out;
char name [12];
in = fopen ("lab.dat", "r");
gets (name);
if ((in1 = fopen (name, "r")) == NULL)
{ printf (" Nevar atvērt failu %s\n", name);
 exit (1);
}
fclose (in); fclose (in1); /* fcloseall (); */
```

69

## Secīgā pieeja failiem programmēšanas valodā C

Secīgā pieeja ir tāda pieeja atmiņai, kuras gaitā datu nolasīšanas vai ierakstīšanas process atkarīgs no datu atrašanās vietas datu vidē un iepriekšējās pieejas tiem

Funkcijas:

• **fputc** (<mainīgais>, <faila rādītājs>) - simbola izvade

• **fprintf** (<faila rādītājs>, <formāti>, arg,...) - datu izvade

• **fputs** (<virknes rādītājs>, <faila rādītājs>) - virknes izvade

• **fflash** (<faila rādītājs>) - datu pārvietošana: buferis - fail

• **fgetc** (<faila rādītājs>) - simbola ievade

• **fscanf** (<faila rādītājs>, <formāti>, ptr,...) - datu ievade

• **fgets** (<virk.rād.>, <garums>, <faila rād.>) - virknes ievade

70

## Failu izveidošanas piemēri programmēšanas valodā C

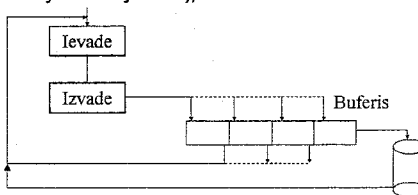
```
include <stdio.h>
main ()
{ FILE *out; int symbols;
 out = fopen ("fails1.dat", "w");
 while ((symbols = getchar ()) != '#')
 fputc (symbols, out);
 fclose (out); }

int kurss = 2; char grupa[] = "DB12";
fprintf (out, "%d. kursa %s grupa\n", kurss, grupa);
/* Failā tiks ievietots: 2. kursa DB12 grupa */
fputs ("Tabula", out); fputs (grupa, out);
```

71

## Buferis datu pārsūtīšanas procesā

Buferis ir operatīvās atmiņas apgabals datu īslaicīgai glabāšanai, tos pārsūtot starp divām ierīcēm (starp atmiņu un ārējo ierīci), lai izlīdzinātu to darbības ātrumus



72

### Datu nolasīšanas no faila piemēri programmēšanas valodā C

```
include <stdio.h>
main ()
{ FILE *in; int simbols;
 if ((in = fopen ("fails1.dat", "r")) != NULL)
 { while ((simbols = fgetc (in)) != EOF)
 { fputc (simbols, stdout); /* putchar (simbols); */
 fclose (out); }
 else printf ("Nevar atvērt failu \"fails1.dat\"\\n"); }

 int kurss; char grupa[20];
 fscanf (in, "%d%s", &kurss, grupa);
 /* Tiks nolasīts: 2. kursa DB12 grupa */ }
```

### Datu nolasīšanas no faila piemēri

(turpinājums)

```
include <stdio.h>
main (int argc, char *argv[])
{ FILE *in;
 if (argc == 1) filecopy (stdin); /* ievade no tastatūras */
 else while (--argc > 0)
 if ((in = fopen (*++argv, "r")) == NULL)
 { printf ("Nevar atvērt failu %s\\n", *argv); break; }
 else { filecopy (in); fclose (in); } }

void filecopy (FILE *in)
{ int simbols;
 while ((simbols = fgetc (in)) != EOF)
 fputc (simbols, stdout); /* putchar (simbols); */ }
```

### Datu nolasīšanas no faila piemēri (2. turpinājums)

```
FILE *in;
char virkne[20]; int kurss;
in = fopen ("fails1.dat", "r");
fscanf (in, "%d", &kurss);
while (fgetc (virkne, 20, in)) != NULL)
 puts (virkne);
```

Fails "fails1.dat": 2 kursa DB12 grupa\\n

Datu nolasīšanas rezultāts: kurss = 2  
virkne = " kursa DB12 grupa"

### Tiešā pieeja failiem programmēšanas valodā C

Funkcija: **fseek** (<faila rādītājs>, <nobīde>, <sākumpunkts>)

Sākumpunkta vērtības: 0 (SEEK\_SET) - faila sākums  
1 (SEEK\_CUR) - tekošā pozīcija  
2 (SEEK\_END) - faila beigas

```
FILE *in; long offset; /* nobīde no sākumpunkta */
in = fopen ("test.txt", "r");
offset = 10L;
if (fseek (in, offset, 0) == 0) putchar(fgetc (in));
fseek (in, -offset, 2); putchar(fgetc (in));
fseek (in, 0L, 0); /* rewind (in); */
```

### Tiešā pieeja failiem (piemēri) programmēšanas valodā C

```
FILE *in; long offset; int k;
in = fopen ("test.txt", "r");
offset = 5L;
k = 1;
while (fseek (in, offset++, 0) == 0 && k < 9)
 { putchar(fgetc (in)); ++k; }
offset = -9L;
while (fseek (in, offset+=9L, 0) == 0)
 putchar(fgetc (in));
fclose (in);
```

### Blokveida ievade-izvade programmēšanas valodā C

- Izvades funkcija  
**fwrite** ( ptr, size, nitem, stream )
  - Ievades funkcija  
**fread** ( ptr, size, nitem, stream )
- ptr** - rādītājs uz **char** tipa objektu (bufera adrese)  
**size** - ieraksta garums ( tips **int** vai **unsigned** )  
**nitem** - ierakstu skaits ( tips **int** )  
**stream** - faila rādītājs ( tips **FILE** )

## Blokveida izvades piemērs programmēšanas valodā C

```
#include <stdio.h>
struct student { char st_apl_numurs;
 char uzvards [25];
 int atzime [5]; }

main ()
{ FILE *out; int garums;
 struct student stud;
 out = fopen ("fails2.dat", "w");
 /* struktūras ievade */
 garums = sizeof (stud);
 fwrite (&stud, garums, 1, out);
 fclose (out); }
```

79

## Papildus funkcijas darbam ar failiem

- **feof** ( <faila rādītājs> )  
Rezultāts (int) = 0, ja nav EOF, un != 0, ja ir EOF
  - **ferror** ( <faila rādītājs> )  
Rezultāts (int) = 0, ja kļūdu nav, un != 0, ja ir kļūdas
  - **clearerr** ( <faila rādītājs> ) Uzstāda kļūdas pazīmi = 0
- ```
FILE *in;
in = fopen ( "fails1.dat", "r" );
while ( !feof(in) )
    putchar ( fgetc (in) );
```

80

Dinamiskā atmiņas iedalīšana programmēšanas valodā C

- **malloc** (<baitu skaits>)
- **calloc** (<elem. skaits>, <baitu skaits katram elem.>)
- **free** (<rādītājs uz iedalītu atmiņu>)

Piemēri:

```
#include <stdlib.h>
#include <alloc.h>
char *malloc (unsigned);
char *calloc (unsigned,int);
int *pi;
float *pf;

pi = (int*) malloc (50);
pf = (float*) malloc (100);
free ( pf );
pf = (float*) calloc (100, sizeof(float));81
```

81

Grafiskie līdzekļi programmēšanas valodā C

Pikselis (*pixel*) ir "minimālais attēla elements, kam rastrgrafikā var tikt patstāvīgi piešķirti tādi raksturojumi, kā, piemēram, krāsa un spilgtums"

Pikseļu skaits uz ekrāna nosaka grafiskās sistēmas izšķirtspēju (*resolution*) $m \times n$, (piemēram, 640x480), kur m - pikseļu skaits rindīņā; n - pikseļu skaits stabīņā

Grafiskie videoadapteri:

- CGA** (Color/Graphics Adapter) - 640x200
- EGA** (Enhanced Graphics Adapter) - 640x350
- VGA** (Video Graphics Array) - 640x480

82

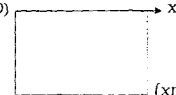
Grafiskās programmas struktūra programmēšanas valodā C

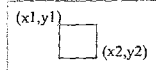
1. Makrokomanda `#include <graphics.h>`
`#include <conio.h>`
`#include <stdlib.h>`
2. Grafiskās sistēmas inicializācija
`int g_driver, g_mode;`
`g_driver = DETECT;`
`initgraph (&g_driver, &g_mode, "ceļš BGI");`
3. Grafisko attēlu izveidošana un darbs ar tiem
4. Grafiskās sistēmas aizvēršana `closegraph ();`

83

Darbs ar pikseļu rastru programmēšanas valodā C

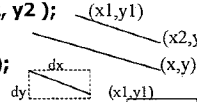


Rastrs ir pikseļu divdimensiju masīvs (matrica)

(0,0)  `xmax = getmaxx ();`
`ymax = getmaxy ();`

`setviewport (x1, y1, x2, y2, p);` 
`setviewport (150, 50, 75, 75, 0);`
 $p=0$ - attēla izvade tikai logā
 $p \neq 0$ - pilnīga attēla izvade

84

Grafiskās pamatfunkcijas programmēšanas valodā C

- Līnija - `line (x1, y1, x2, y2);` 
 - `lineto (x, y);`
 - `linerel (dx, dy);`
- Taisnstūris `rectangle (x1, y1, x2, y2);` 
- Ripīga līnija `circle (x, y, r);` 
- Elipse `ellipse (x, y, a1, a2, rx, ry);`
- Daudzstūris `drawpoly (n, masivs);`
`int m[10] = {0,0, 100,0, 200,50, 100,50, 0,0}; drawpoly (4,m);`

Darbs ar krāsām programmēšanas valodā C

- fona krāsa `setbkcolor (krāsa);`
 - līniju krāsa `setcolor (krāsa);`
 - pikseļa izvade `putpixel (x, y, krāsa);`
 - aizpildīšana `setfillstyle (šablons, krāsa);`
- | Šablons | |
|---------|-----------------|
| 0 | EMPTY_FILL |
| 1 | SOLID_FILL |
| 2 | LINE_FILL |
| 3 | LTSLASH_FILL |
| 4 | SLASH_FILL |
| 5 | BRSLASH_FILL |
| 6 | LTBRSLASH_FILL |
| 7 | HATCH_FILL |
| 8 | XHATCH_FILL |
| 9 | INTERLEAVE_FILL |
| 10 | WIDE_DOT_FILL |

Grafiskās programmas piemērs programmēšanas valodā C

```
#include <stdio.h>
#include <graphics.h>
main ( )
{ int gdriver, gmode;
  int i, j;
  gdriver = DETECT;
  initgraph ( &gdriver, &gmode, "c:\\Borlandc\\BGI");
  setbkcolor ( LIGHTGRAY );
  for ( i = 0; i <= getmaxx ( ); i += 10 )
    for ( j = 0; j <= getmaxy ( ); j += 10 )
      putpixel ( i, j, RED );
  closegraph (); }
```

Grafikas lietošanas piemēri programmēšanas valodā C

```
int x = 100, y = 100, r = 75,
    x1 = 350, y1 = 200, r1 = 150;
setbkcolor ( BLUE ); setcolor( LIGHTBLUE );
setfillstyle ( SOLID_FILL, WHITE );
circle ( x, y, r );
setfillstyle ( LTSLASH_FILL, 15 );
ellipse ( x1, y1, 0, 360, r1, r );
setfillstyle ( WIDE_DOT_FILL, RED );
rectangle ( 250, 50, x+x1, y );
cleardevice ( );
```

Grafiskā attēla saglabāšana programmēšanas valodā C

- Saglabāšana atmiņā
`getimage (x1, y1, x2, y2, atmiņas adrese);`
 - Nepieciešamā atmiņas apjoma noteikšana
`imagesize (x1, y1, x2, y2);`
 - Izvade uz ekrāna
`putimage (x, y, atmiņas adrese, operācija);`
- Operācija: 0 COPY_PUT
1 XOR_PUT
2 OR_PUT
3 AND_PUT
4 NOT_PUT

Grafiskās programmas piemērs programmēšanas valodā C (2)

```
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#define X1 50
#define Y1 50
#define Rx 100
#define Ry 75
main ( )
{ int gdriver, gmode, code, size; void *buf; FILE *fp;
  gdriver = DETECT;
  initgraph ( &gdriver, &gmode, "c:\\Borlandc\\BGI");
  code = graphresult ( ); if ( code != grOk )
  { printf ("Iniciālizācijas kļūda\n"); exit(1); }
```

Grafiskās programmas piemērs programmēšanas valodā C (2+)

```
setbkcolor (WHITE);  
setfillstyle (SOLID_FILL, GREEN );  
ellipse ( X1+Rx, Y1+Ry, 0, 360, Rx, Ry );  
size = imagesize ( X1, Y1, X1 + 2*Rx, Y1 + 2*Ry);  
buf = malloc ( size );  
getimage ( X1, Y1, X1 + 2*Rx, Y1 + 2*Ry, buf );  
fp = fopen ("ellipse.dat", "w+");  
fwrite ( buf, size, 1, fp );    getch ( );  
fread ( buf, size, 1, fp );  
putimage ( 300, 250, buf, COPY_PUT );  
getch ( );  
closegraph ();  
}
```

91