Programavimas C kalba

Mokomoji priemonė
Elektronikos specialybės studentams

Vytautas Vyšniauskas

2008.01.28 09:26

Šiaulių Universitetas, 2007

Turinys

Įvadas

Žmonija nuo seno skaičiuoja ir skaičiavimams palengvinti naudoja įvairias priemones. Ties metodais kuriuose buvo naudojami akmenukai ,kriauklelės ir t.t. mes neapsistosime. Padarę didžiulį šuolį žmonijos istorijoje, grįšime nepilną šimtą metų atgal ir trumpai prisiminsime kaip buvo programuojamos pirmosios elektroninės skaičiavimo mašinos.

Kaip ir šiuolaikiniai kompiuteriai anų laikų skaičiavimo mašinų pagrindinės dalys buvo centrinis skaičiavimo įrenginys (CPU – **C**entral **P**rocessing **U**nit), dabar jis vadinamas centriniu procesoriumi arba tiesiog procesoriumi, atmintis (memory), ir įvesties (input) bei išvesties (output) įrenginiai. Tiesą sakant šiuolaikiniame personaliniame kompiuteryje dažniausiai yra ne vienas procesorius arba mikrovaldiklis, jie visi veikia pagal tam tikras programas ir vykdo tamtikras kompiuterio funkcijas. Kai kuriuos iš jų taip pat galima programuoti, o kai kurie turi gamybos metu įrašytas programas ir jų pakeisti nebegalima. Pavyzdžiui 3D klasės vaizdo procesoriai gali apdoroti vaizdą keliasdešimt kartų greičiau už kompiuterio centrinį procesorių. Todėl kompiuterinių žaidimų programos yra rašomos specialiai tokiems procesoriams. Tokie vaizdo procesoriai sukuria beveik realų vaizdą kompiuterio ekrane.

Kad procesorius „žinotų“ kokius veiksmus ir su kuo reikia atlikti reikalinga programa. Programa yra instrukcijų seka, kuri yra skirta centriniam procesoriui. Pirmosios skaičiavimo mašinos neturėjo programavimo kalbų ir buvo programuojamos mašininiais kodais. Štai tokio kodo pavyzdys.

```
56 68 A0 D9 72 10 50 52 BA 04 10 00 00 B9 5B 01
00 00 E8 7F A4 01 00 8B 7E 1C BA 01 00 00 00 8A
4F 08 D3 E2 F6 C2 06 74 31 8B 4F 14 8B 51 14 85
D2 74 27 8B 50 1C 8B 4A 14 8B 51 08 8B 7A 0C 85
FF 75 10 89 41 14 C7 46 04 45 01 00 00 8B C6 5F
5E 5B C3
```

Tokiu būdu rašyti programas būdavo labai sudėtinga. O didžiausia problema, kad buvo sukurta daugybė skaičiavimo mašinų, kurių mašininės kalbos kodai skyrėsi. Todėl programuotojui norint programuoti kitokią mašiną, reikėdavo mokytis kitos mašinos komandų sistemą.

Dalinai sprendimas buvo rastas sukūrus Asemblerį. Štai prieš tai buvęs mašininis kodas asemblerio kalboje matomas trečiame ir ketvirtame stulpelyje. Šioje kalboje jau naudojami žodžiai, kurie dažniausiai yra anglų kalbos žodžių trumpiniai.

```
107150DA                L107150DA:
107150DA  56                push   esi
107150DB  68A0D97210        push   SUB_L1072D9A0
107150E0  50                push   eax
107150E1  52                push   edx
107150E2  BA04100000        mov    edx,00001004h
107150E7  B95B010000        mov    ecx,0000015Bh
107150EC  E87FA40100        call   SUB_L1072F570
107150F1  8B7E1C            mov    edi,[esi+1Ch]
107150F4  BA01000000        mov    edx,00000001h
107150F9  8A4F08            mov    cl,[edi+08h]
107150FC  D3E2              shl    edx,cl
107150FE  F6C206            test   dl,06h
10715101  7431              jz     L10715134
10715103  8B4F14            mov    ecx,[edi+14h]
10715106  8B5114            mov    edx,[ecx+14h]
10715109  85D2              test   edx,edx
1071510B  7427              jz     L10715134
1071510D  8B501C            mov    edx,[eax+1Ch]
10715110  8B4A14            mov    ecx,[edx+14h]
10715113  8B5108            mov    edx,[ecx+08h]
10715116  8B7A0C            mov    edi,[edx+0Ch]
10715119  85FF              test   edi,edi
1071511B  7510              jnz    L1071512D
1071511D  894114            mov    [ecx+14h],eax
10715120  C7460445010000    mov    dword ptr [esi+04h],00000145h
```

```
10715127  8BC6            mov    eax,esi
10715129  5F              pop    edi
1071512A  5E              pop    esi
1071512B  5B              pop    ebx
1071512C  C3              retn
```

Tačiau tai tik dalinai palengvino programuotojų darbą, nes skirtingos mašinos turėjo ne vienodą asemblerio komandų rinkinį, nors didžioji dalis komandų buvo tos pačios.

Augant skaičiavimo mašinų skaičiui, atsirado poreikis turėti tas pačias programas įvairiose mašinose. Kaip buvo minėta, asembleris tik dalinai pašalino skirtingų procesorių instrukcijų tarpusavio skirtumus. Todėl buvo sukurtos, taip vadinamos, aukšto lygio kalbos, kuriose procesoriaus instrukcijų nebebuvo. Tačiau reikėjo turėti specialią programą kompiliatorių, kuri mokėjo „paversti“ programos tekstą mašininiu kodu.

Viena iš pirmųjų aukšto lygio kalbų, C (tariama „sy“) kalba, buvo sukurta praėjusio šimtmečio 7-tame dešimtmetyje. Gal todėl, kad ji buvo viena iš pirmųjų, gal todėl, kad ją sukūrė tikrai genialūs žmonės, tačiau C kalba papildyta objektinio programavimo galimybėmis (C++), labai plačiai naudojama iki šiol.

Kokie yra C kalbos privalumai?

- Efektyvi programavimo kalba, leidžianti efektyviai išnaudoti kompiuterio resursus. C kalba parašytos programos yra kompaktiškos, greitai vykdomos centrinio procesoriaus. Kai kurioms kitoms kalboms reikalinga virtuali mašina.
- C kalboje galima naudoti asemblerio kodą (inline assembler), tuo dar labiau padidinant jos efektyvumą, o taip pat priartinant prie žemo lygio kalbos.
- Galinga ir lanksti kalba, kuria parašytos beveik visos operacinės sistemos, arba bent jau didžioji operacinės sistemos dalis. Pavyzdžiui UNIX, Windows, MAC OS, LINUX operacinės sistemos parašytos beveik vien C ir C++ kalba, bei naudojant nedideles funkcijas parašytas asembleriu.
- Turbūt visi žino, kad LINUX operacinė sistema yra pritaikyta nuo personalinių kompiuterių iki didžiųjų skaičiavimo mašinų. Tai yra galima todėl, kad C kalboje rašytas programas su labai nedideliais pakeitimais, arba net ir visai be pakeitimų galima perkelti į kitas kompiuterių sistemas. Tai yra dar viena C kalbos savybė – pernešamumas.

C kalba yra programuojami ir šiuolaikiniai mikrovaldikliai, kurie plačiai naudojami laikrodžiuose, telefonuose, foto ir kino kamerose, šaldytuvuose ir skalbimo mašinose, automobiliuose, lėktuvuose ir kosminiuose laivuose.

Dauguma girdėjote apie kitas programavimo kalbas, arba mokėtės programuoti: Paskal, Delfi, Java, Perl ir t.t. Kam reikalinga tokia programavimo kalbų įvairovė? Gal pakanka C kalbos? Kompiuteriais yra sprendžiamos labai įvairios problemos, nuo teksto rašymo, ekonominių skaičiavimų iki astronominių skaičiavimų. Yra programavimo kalbų, kuriomis labai lengva rašyti programas tekstų apdorojimui, kitos kalbos skirtos interneto svetainių kūrimui, kitos inžineriniams ir moksliniams skaičiavimams, kitos astronominiams skaičiavimams. Kam žmogui reikia mokėti kalbą, kuria galima atlikti sudėtingus matematinius skaičiavimus, jei jis nori sukurti puslapį internetui? Taigi programavimo kalbų įvairovė yra reikalinga. Juo labiau, kad viena kalba parašyta programos dalis sudaranti dešimtis eilučių, kitoje kalboje, ta pati programos dalis, yra vos kelios eilutės.

1 Simbolių kodavimas ir ASCII kodų lentelė

Kompiuteris dirba tik su skaičiais, tad norint turėti kompiuteryje raides ir kitus simbolius, reikėjo sukurti sistemą, kaip simbolius pakeisti skaičiais. Tad Amerikos standartų institutas 1963 metais pristatė 7 bitų ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) kodų lentelę pavaizduotą 1 paveikslas. ASCII yra tariamas kaip „ask-ky“ (angliškai „ask-key).

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

1.1 pav. ASCII kodų lentelė

Simbolių atvaizdavimui buvo naudojamas 8 bitų žodis, iš kurių buvo naudojami tik 7 bitai. Šis standartą naudojamas iki šiol. Čia yra ir kodai simbolių atvaizdavimui HTML puslapiuose. Pirmieji 32 simboliai buvo panaudoti kaip valdymo simboliai, nes tuo metu įvesties/išvesties įrenginiu dažniausiai buvo telegrafo aparatas (teletype).

Be lotyniškų raidžių reikėjo ir kitokių simbolių, todėl 1968 metais ASCII lentelė buvo išplėsta kodais pavaizduotais 2 paveiksle. Čia matome graikiškas, fokiškas ir kitokias raides, lentelių, grafinius ir matematinius simbolius.

128	Ç	144	É	161	í	177	▒	193	┴	209	╤	225	ß	241	±				
129	ü	145	æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥				
130	é	146	Æ	163	ú	179	│	195	├	211	╙	227	π	243	≤				
131	â	147	ô	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠				
132	ä	148	ö	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡				
133	à	149	ò	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷				
134	å	150	û	167	º	183	╖	199	╟	215	╫	231	τ	247	≈				
135	ç	151	ù	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°				
136	ê	152	_	169	_	185	╣	201	╔	217	┘	233	Θ	249	·				
137	ë	153	Ö	170	¬	186	║	202	╩	218	┌	234	Ω	250	·				
138	è	154	Ü	171	½	187	╗	203	╦	219	▉	235	δ	251	√				
139	ï	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	_				
140	î	157	¥	173	¡	189	╜	205	═	221	▌	237	ϕ	253	²				
141	ì	158	_	174	«	190	╛	206	╬	222	▐	238	ε	254	■				
142	Ä	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255					
143	Å	160	á	176	▒	192	└	208	╨	224	α	240	≡						

Source: www.LookupTables.com

1.2 pav. ASCII lentelės išplėtimas

ASCII lentelės išplėtime yra 128 simboliai, kurių kodai yra nuo 128 iki 255. Šiuo metu galioja ASCII lentelės aprašytos dokumentuose *ISO-14962-1997* ir *ANSI-X3.4-1986(R1997)*. Tačiau

lentelės išplėtimas (128 – 255) yra ne visur vienodas. Pavyzdžiui Microsoft Windows ® naudoja kitokią kodų lentelę. Tikriausiai visi susidūrėte, kai interneto puslapis rodo nesuprantamus simbolius vietoje lietuviškų raidžių, o rusiški puslapiai visiškai neperskaitomi. Čia kaltas simbolių kodavimas ir didžiulė kodų lentelių įvairovė, bei puslapio kūrėjas, kuris ne viską padarė, kad puslapis veiktų teisingai.

2 Programavimo C kalba pagrindai

Pradėdami mokytis programuoti C kalba, pažiūrėkime į paprasčiausią programos tekstą.

```
int main() {}
```

arba

```
int main()
{

}
```

Šią programą galima sukompiliuoti ir gauti vykdomą failą, tačiau ši programa nieko nedaro. Programa, C kalboje, yra visada aprašoma kaip funkcija *main()*. Programos veiksmai yra apgaubiami figūriniais {} skliausteliais. Kaip tai padaryta sekančioje programoje.

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

int main(int argc, char* argv[]) {
    printf("Pirmasis programos pavyzdys C kalba. \n");
    printf("\n");

    /* sekanti eilutė neleidžia užsibaigti programai
       ir užsidaryti consolės langui, kai programuojama
       Windows OS arba kitoje grafinėje.
    */

    system("Pause");
    return (0);
}
```

```
Pirmasis programos pavyzdys C kalba.

Press any key to continue . . .
```

Matome, kad C kalboje yra naudojami anglų kalbos žodžiai, anglų kalbos žodžių trumpiniai, arba sudurtiniai žodžiai. Beje visuose programavimo kabose yra naudojami anglų kalbos žodžiai ir terminai.

Tiems kas dar nemoka programuoti nelabai aišku kaip ši programa veikia. Apie viską bus aprašyta detaliai, o dabar apie tai kas bus dažniausiai sutinkama programų pavyzdžiuose.

Programa yra vykdoma iš kairės į dešinę ir iš viršaus į apačią. Taip kaip esame įpratę skaityti. Vėliau matysime, kad programa gali peršokti kelias eilutes į apačią ar į viršų, bet vėliau vis vien yra vykdoma iš kairės į dešinę ir iš viršaus į apačią.

Pirmosios dvi programos eilutės yra antraščių (header) failų įtraukimas į programos tekstą. Matysime, kad būtent šios antraštės ir yra dažniausiai įtraukiamos į programos tekstą. Antraščių failuose yra aprašytos, jau seniau sukurtos funkcijos, kurios yra naudojamos programoje. Funkcija yra programos dalis, kuri gali būti naudojama ir kitose programose be pakeitimo. Tokios „naudingos“ funkcijos yra surenkamos į bibliotekas ir joms sudaromi antraščių failai, kuriuose yra

funkcijų prototipai arba deklaracijos. Kaip jau buvo minėta, programa yra *main()* funkcija, o figūriniuose skliausteliuose aprašomi programos veiksmai. Pirmasis programos veiksmas yra funkcija *printf()*, kuri ekrane atspausdina užrašą esantį kabutėse. Daugiau apie funkciją printf() galima pasiskaityti priede C. Sekantis veiksmas yra dar vienas spausdinimo funkcijos iškvietimas, kuriame nurodoma atspausdinti simbolį „nauja eilutė“. Trečiasis programos veiksmas yra operacinės sistemos komandos „Pause“ iškvietimas, kuriam yra naudojama C kalbos funkcija *system()*. Ketvirtasis veiksmas yra *return (0)*, kuris reiškia, kad funkcija, šiuo atveju programa, užsibaigia grąžindama reikšmę 0. Operacinėse sistemose ir C kalboje yra priimta, kad kai programa baigiasi sėkmingai, ji grąžina kodą 0. Jei programoje kyla klaida, progama grąžina klaidos kodo numerį.

2.1 Sintaksė ir terminai

C kalba sudaryta iš įvairių į žodžius panašių vienetų, vadinamų **žodžiais** arba **tokenais**. Tokenų grupės sudaro sakinius, frazes, išraiškas ir kitas kalbos dalis. Kompiliatorius skaitydamas programos tekstą suskirsto jį į tokenus ir tarpus. Tarpas (whitespace) yra bendras vardas eilei ASCII simbolių ir C kalbos konstrukcijų, tai: tarpelis, horizontali ir vertikali tabuliacija, naujos eilutės simbolis ir komentarai.

Komentarai C kalboje yra dviejų rūšių: turintys atidarymo (/*) ir uždarymo (*/) simbolius ir komentaras prasidedantis (//) ir pasibaigiantis eilutės pabaigos simboliu. Komentarai naudojami programos paaiškinimams užrašyti. Programa su komentarais yra aiškesnė, lengviau suprantama. Komentarai yra naudingi ne tik kitam žmogui, nagrinėjančiam programą, bet ir programos autoriui, nes praėjus laikui, viską prisiminti yra neįmanoma, o gerai parašyti komentarai žymiai sutrumpina laiką, kurio reikia programai suprasti. Šiuolaikinės programos yra sudėtingos, jas sudaro šimtai, o kartais ir šimtai tūkstančių eilučių, bei dešimtys ar šimtai failų. Tokias programas kuria kolektyvai todėl komentarai yra ne tik pageidaujami, bet tiesiog privalomi. Štai keletas komentarų pavyzdžių:

```
int  counter1;          /* Skaitiklis bandymų skaičiui*/
long counter2;          // Laikas sekundėmis
```

Komentaras gali apimti kelias eilutes:

```
/* ------------------------------------------------------------
      Šiaulių universitetas
            Technologijos fakultetas
                  Elektronikos katedra
------------------------------------------------------------*/
```

Komentaras /* ... */ negali būti naudojamas tokio pat komentaro viduje, pavyzdžiui šioje programoje yra klaida.

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

int main(int argc, char *argv[])
{
/* ------------------------------------------------------------
      Siauliai University
/*          Faculty of Technology   */
                  Electronics department
------------------------------------------------------------*/

  system("PAUSE");
  return 0;
}
```

Kadangi komentaras, prasidedantis simboliu /*, baigiasi, kai kompiliatorius randa pirmąjį komentaro užbaigimo simbolį */.

Esant reikalui panaudoti komentarą viduje /* ... */ komentaro, galima naudoti // tipo komentarą.

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

int main(int argc, char *argv[])
{
/* ------------------------------------------------------------
      Siauliai University
//          Faculty of Technology
                  Electronics department
------------------------------------------------------------*/

  system("PAUSE");
  return 0;
}
```

Toks poreikis gali atsirasti tik programos derinimo metu. Kadangi komentarą kompiliatorius „išmeta“ iš programos, todėl kai reikia kad kompiliatorius „neimtų“ kažkurios programos teksto dalies kompiliuojant, ją galima užkomentuoti.

Komentarams rašyti labai patogu naudoti skaitmeninę klaviatūrą (dešinėje klaviatūros pusėje). Pabandykite /**/ arba //.

3 Vardai

C kalboje vardus turi kintamieji, konstantos, funkcijos ir kitos kalbos dalys arba objektai. Vardai turi būti unikalūs tam tikroje programos ar funkcijos dalyje. Dažniausiai vardai būna unikalūs visoje programoje ar funkcijoje. Vardas turi prasidėti lotyniškos abėcėlės raide arba pabraukimo (underscore _) simboliu, visi kiti vardo simboliai gali būti parinkti iš šių grupių:

a ... z	mažosios lotyniškos raidės nuo **a** iki **z**
A ... Z	didžiosios lotyniškos raidės nuo **A** iki **Z**
0 ... 9	skaičiai nuo **0** iki **9**)
_	pabraukimas (underscore)

Keletas teisingų ir neteisingų C kalbos vardų pavyzdžių:

```
// Teisingi vardai
int             i, j, i_var, L1, skaicius;
long            total, viso;
bool            Out_of_Memory;
short int       VAR;
char            Name[80], MyData[10];
unsigned int    _2end, B52;

// neteisingi vardai
double          2Pi=2*M_PI;
char            sąrašas[10], *pradžia;
bool            žymė;
unsigned char   raidė;
float           #capacity;
char            E@mail[20];
```

C kalbos standartas varduose neleidžia naudoti lietuviškų ir kitokių užsienietiškų raidžių. Nors kai kurie kompiliatoriai nekontroliuoja kokie simboliai naudojami vardams, bet geriau naudoti leistinus simbolius.

Vardus reikėtu parinkti prasmingus, tada programa tampa aiškesnė ir sugaištama mažiau laiko jos veikimui suprasti. Yra ir „nusistovėję“ vardai t.y. tokie vardai, kuriuos naudoja dauguma

programuotojų. Pavyzdžiui ciklų kintamųjų vardai dažniausiai yra *i, j*, vienas simbolis *c* ir t.t. Tai taip pat palengvina programos supratimą.

4 Skyrybos ženklai

Skyrybos ženklai (punctuators), kai kada dar vadinami skirtukais (separators) C kalboje yra tokie:

```
[] () {} , ; : ... * = #
```

4.1 Laužtiniai skliausteliai

Laužtiniai skliausteliai (brackets) [ ] pažymi vienmačius ir daugiamačius masyvus

```
char ch;                        /* simbolis */
char str[] = "Masyvas";         /* vienmatis masyvas */
char mat[3][4];                 /* dvimatis (3 X 4) masyvas */
str[3];                         /* 4-sis masyvo str[] elementas
                                   Simbolis "y" */
```

4.2 Lenktiniai skliausteliai

Lenktiniai skliausteliai (parentheses) ( ) yra naudojami grupuoti išraiškas, izoliuoti sąlygines išraiškas, pažymėti funkcijos iškvietimą, jais taip pat apgaubiami funkcijos parametrai:

```
d = c * (a + b);              /* pakeičiama veiksmų atlikimo tvarka */
if (d == z) ++x;              /* apgaubiama sąlygos išraiška */
func ();                      /* funkcijos be argumentų iškvietimas */
int (*fptr) ();               /* funkcijos rodyklės deklaracija */
fprt = func;                  /*  nėra () reiškia rodyklę į funkciją */
void func2 (int n, char c);   /* funkcijos su argumentais deklaracija */
```

Lenktiniai skliausteliai yra rekomenduojami makro procedūrose, kad išvengti klaidų jas išplečiant.

```
#define CUBE(x) ((x) * (x) * (x))
```

Tipo pakeitimo operacijose (typecast) naujam tipui priskirti:

```
int a;
long b;
a = (int)b;
```

4.3 Figūriniai skliausteliai

Figūrinai skliausteliai (braces) { } nurodo mišrių veiksmų ar veiksmų grupės pradžią ir pabaigą, kitaip tariant apbaubia veiksmų grupę:

```
if  (d = = x)
{
      func();
      ++x;
}
```

Atidarantis skliaustelis pažymi veiksmų grupės pradžią, o uždarantis skliaustelis grupės pabaigą, todėl po jo kabliataškis (;) nereikalingas. Išimtį sudaro struktūrų ar klasių deklaracija.

Sekančiame pavyzdyje po sąlygos kabliataškis niekada nerašomas:

```
if (sąlyga)              // kabliataškis niekada nerašomas
      {};                /* nereikalingas kabliataškis */
else
```

Figūriniai skliausteliai taip pat apgaubia funkcijos veiksmus. Funkcijos veiksmas prasideda už atidarančiojo skliaustelio ir baigiasi prieš uždarantįjį skliaustelį. Skliaustelių trūkumas ar perteklius taip kaip ir kabliataškiai yra gan dažna programavimo klaida. Nors šiuolaikiniai kompiliatoriai gerai seka skliaustelius, tačiau pranešimas apie klaidą ne visada būna aiškus, o kartais apie skliaustelius net nekalbama.

Programuojant reikia laikytis vienos taisyklės: – jei skliaustelį atidarei, reikia ir uždaryti. Tam labai pasitarnauja tvarkingas programavimas dar vadinamas programavimo stiliumi, kai veiksmų grupes atitraukiamos nuo krašto atitinkamai atitraukiami skliausteliai, vienodai naudojami tarpeliai ir t.t.

4.4 Kablelis

Kablelis (comma) (,) atskiria funkcijos argumentus, vienodo tipo kintamųjų vardus ir kintamųjų reikšmes:

```
void func (int n, float f, char ch);
int i, j, k, l, m;
myStr m={1, "Boing", 747, "  ", 426};
long laray[7]={1, 7, 4, 7, 4, 2, 6};
```

4.5 Kabliataškis

Kabliataškis (semicolon) (;) yra reiškinio ar išraiškos pabaigos ženklas. Kiekvienas C kalbos reiškinys ar išraiška visada baigiasi (;) įskaitant ir „tuščią operatorių“, kuris žymimas (;).

```
a + b;       // išraiška a+b išsprendžiama, bet reikšmė prarandama
a++;         // a reikšmė padidinama vienetu
;            // tuščias operatorius

lambda = 1/(max (sqrt(abs(f1)), sqrt(abs(f2)));
```

Kabliataškis dažnai naudojamas kaip tuščias operatorius cikluose laiko intervalų sudarymui. Tuščias operatorius yra sutransliuojamas į asemblerio **NOP** (**N**o **OP**eration) ir yra vykdomas per vieną procesoriaus taktą. Jei parašyti kelis kabliataškius, bus atliekami keli tušti operatoriai ir tai užtruks tiek pat procesoriaus taktų. Tokie ciklai naudojami laiko intervalų formavimui, dažniausiai senesniuose mikroprocesoriuose ir mikrovaldiliuose, kurie neturi arba turi nepakankamai laikmačių (taimer).

```
for (i = 0; i < 1000; i++)
{
      ;;;;;;;;    // tuščias operatorius
}
```

Praleistas kabliataškis išraiškos gale yra viena dažniausiai pasitaikančių klaidų, tarp pradedančiųjų, ir ne tik, programuotojų.

4.6 Dvitaškis

Dvitaškis (colon) (:) parodo žymę programoje t.y. pažymi vietą į kurią gali būti perduotas programos valdymas.

```
start :
    x = 0;
    . . .
    if (x >= a) goto start;
    switch (a) {
        case 1:   puts("Vienas");
            break;
        case 2:   puts("Du");
            break;
        . . .
        default puts("Nei viena iš aukščiau esančių reikšmių");
            break;
    }
```

Operatorius **goto label:**, programuojant C kalba yra naudojamas labia retai. Jo naudojimas yra laikoma prastu programavimo stiliumi, nes gali sukelti begalinį ciklą, kurį sunku aptikti.

4.7 Daugtaškis

Daugtaškias (ellipsis) (…) trys taškai be tarpų. Daugtaškis naudojamas pažymėti formaliems parametrams funkcijų prototipuose. Jis pažymi kad funkcija turi kintamą parametrų skaičių arba kad kintamieji yra skirtingų tipų.

```
void func (int n, char ch, ...);
```

Ši deklaracija rodo, kad funkcija **func** yra aprašyta taip, kad ją iškviečiant būtina perduoti du argumentus **int** ir **char** tipo, tačiau argumentų gali būti ir daugiau. Jeigu *stdio.h* antraščių faile susirasite *printf* ir *fprintf* funkcijų deklaracijas jos bus pavyzdžiui tokios:

```
_CRTIMP int __cdecl     fprintf (FILE*, const char*, ...);
_CRTIMP int __cdecl     printf (const char*, ...);
```

4.8 Žvaigždutė

Žvaigždutė (asterisk) (*) priklausomai nuo to kur yra naudojama gali turėti skirtingas reikšmes.

Žvaigždutė naudojama kaip daugybos ženklas:

```
float r, l;
l = 2 * 3.14 *  r;
```

Žvaigždutė gali būti naudojama rodyklių į kintamuosius deklaracijai:

```
char *char_ptr;          /* rodyklė į char tipo kintamąjį */
char *argv[];            /* rodyklė į char tipo masyvą */
FILE *input;             /* rodyklė į struktūrą FILE */
```

Žvaigždutė gali būti naudojama įvairaus gylio rodyklėms deklaruoti:

```
int     **int_ptr;       /* rodyklė į int tipo rodyklę
                            arba rodyklė į dvimačio masyvo elementą */
double ***double_ptr;    /* rodyklė į rodyklę į double tipo rodyklę
                            arba rodyklė į trimačio masyvo elementą */
```

Žvaigždutė naudojama ir reikšmei iš rodykle aprašyto kintamojo gauti:

```
int i = *int_ptr;
```

Daugiau apie žvaigždutės naudojimą pažymint rodykles skaitykite skyrelyje Rodyklės.

4.9 Lygybės ženklas

Lygybės ženklas (equal, initialize) (=) naudojamas inicializuoti (suteikti reikšmes) kintamiesiems ir atskiria kintamojo deklaraciją nuo kintamojo reikšmių. Kitaip negu matematikoje, kur ženklas „=“ reiškia kad reiškinys kairėje lygybės pusėje yra lygus reiškiniui dešinėje pusėje, C kalboje lygybė reiškia, kad kintamajam kairėje pusėje yra priskiriama (suteikiama) dešinėje pusės esančio reiškinio reikšmė.

```
char array[5] = {0, 1, 2, 3, 4};     /* masyvo elementams priskiriamos
                                        reikšmės */
int x = 5;                           // x priskiriama reikšmė 5
```

C kalbos standartas nustato, kad funkcijose pirmiausia yra deklaruojami kintamieji ir tik po to galima rašyti programos veiksmus. Kaip ir visur yra keletas išimčių. Šiuo atžvilgiu C++ kalboje yra daugiau laisvės, čia yra vienintelis reikalavimas, kad kintamasis turi būti deklaruotas anksčiau nei jis panaudojamas programoje.

Funkcijų deklaracijoje lygybė naudojama pradinių reikšmių (*default values*) nustatymui:

```
int func(int n = 0) { ... }
```

Lygybė taip pat naudojama kaip priskyrimo operacija išraiškose:

```
a = b + c;
i_ptr = calloc(sizeof(int) * 100);
```

Priskyrimo operacija atliekama po to, kai apskaičiuojama dešinėje pusėje esančio reiškinio reikšmė.

5 Priešprocesoriaus komandos

Priešprocesoriaus komandos (*preprocessor commands*) yra pažymimos numerio ženklu # (įvairiose šalyse jis yra vadinamas skirtingai: pound sign, sharp, ceche. Programavimo literatūroje sutinkami visi) žymi preprocesoriaus direktyvą, bet gali būti naudojamas, kaip paprastas simbolis tekstinėse eilutėse ir komentaruose. Preprocesoriaus direktyvai atpažinti, kai kuriuose kompiliatoriuose, šis ženklas visada turi būti pirmas simbolis eilutėje. Preprocesoriaus direktyvos dažniausiai būna programos teksto pradžioje, tačiau gali būti naudojamos bet kurioje programos vietoje. Pagrindinės preprocesoriaus direktyvos yra:

```
# (null directive)   #ifdef
#define              #ifndef
#elif                #import
#else                #include
#endif               #line
#error               #pragma
#if                  #undef
```

5.1

null direktyva. Eilutė, kurioje yra tik šis simbolis yra ignoruojama

5.2 #define, #undef

#define direktyva nustato makro arba konstantas. Makro yra mechanizmas leidžiantis pakeisti vieną tokeną kitu arba keliais, su argumentais ar be argumentų.

Sintaksė:

```
#define makro-identifikatorius <tokenų seka>
```

Pavyzdžiui tokia programa:

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

#define HI "Have a nice day!"
#define Ciao "all the best!"
#define ABS(x) ((x) < 0) ? -(x) : (x)
#define Pi() (3.1415927)
#define _E_   2.7182818

int main(int argc, char *argv[])
{
    int var=10, nvar=-15;
    printf(HI);
    printf("\n\n");
    printf("ABS(%+d)=%d\n", var, ABS(var));
    printf("ABS(%+d)=%d\n\n", nvar, ABS(nvar));
    printf("Pi()=%1.16f\n",Pi());
    printf("_E_ =%1.16f\n\n",_E_);
    printf("M_PI=%1.16f\n",M_PI);
    printf("M_E =%1.16f\n\n",M_E);
    printf("%s\n",Ciao);
    printf("\n");
    system("Pause");
    return(0);
}
```

```
Have a nice day!

ABS(+10)=10
ABS(-15)=15

Pi()=3.1415926999999999
_E_ =2.7182818000000002

M_PI=3.1415926535897931
M_E =2.7182818284590451

all the best!

Press any key to continue . . .
```

Kiekvieną **makro** identifikatorių preprocesorius „išskleidžia“ su galimais tuščiais argumentais. Pavyzdžiui vietoje konstantos *Hi* bus įrašyta *"Have a nice day!"*. Štai kaip atrodo prieš tai buvusios programos dalis po preprocesoriaus:

```
. . .
int main(int argc, char *argv[])
{
    int var=10, nvar=-15;
```

```
        printf("Have a nice day!");
        printf("\n\n");
        printf("ABS(%+d)=%d\n", var, ((var) < 0) ? -(var) : (var));
        printf("ABS(%+d)=%d\n\n", nvar, ((nvar) < 0) ? -(nvar) : (nvar));
        printf("Pi()=%1.16f\n",(3.1415927));
        printf("_E_ =%1.16f\n\n",2.7182818);
        printf("M_PI=%1.16f\n",3.14159265358979323846);
        printf("M_E =%1.16f\n\n",2.71828182845904523536);
        printf("%s\n","all the best!");
        printf("\n");
        system("Pause");
        return(0);
    }
```

Direktyva **#undef** "atšaukia" identifikatoriaus nustatymą.

Pavyzžiui:

```
#undef HI
```

reiškia, kad identifikatorius HI nebeegzistuoja. Bandant jį panaudoti kompiliatorius aptiks klaidą.

Identifikatorius nustatytas su direktyva **#define** egzistuoja iki tol kol jis nebus „atšauktas" t.y. panaudota **#undef** direktyva.

5.3 #ifdef, #ifndef

Sąlyginės kompiliacijos direktyva leidžianti kompiliuoti programos tekstą priklausomai nuo to ar identifikatorius nustatytas ar ne.

Sintaksė:

```
#ifdef identifikatorius
#else or #elif
#endif

#ifndef identifikatorius
#else or #elif
#endif
```

#ifdef – reiškia „jei nustatyta" (if defined)
#ifndef – reiškia „jei nenustatyta" (if not defined)

Šios direktyvos dažnai naudojamos antraštėse, kad antraštė nebūtų įtraukiama pakartotinai. Pavyzdžiu antraščių faile *conio.h*:

```
#ifdef
/*
 * conio.h
 * This file has no copyright assigned and is placed in the Public Domain.
 * This file is a part of the mingw-runtime package.
 * No warranty is given; refer to the file DISCLAIMER within the package.
 *
 * Low level console I/O functions. Pretty please try to use the ANSI
 * standard ones if you are writing new code.
 *
 */

#ifndef     _CONIO_H_
#define     _CONIO_H_

/* All the headers include this file. */
#include <_mingw.h>
```

```
#ifndef RC_INVOKED

#ifdef      __cplusplus
extern "C" {
#endif

_CRTIMP char* __cdecl   _cgets (char*);
_CRTIMP int __cdecl     _cprintf (const char*, ...);
_CRTIMP int __cdecl     _cputs (const char*);
_CRTIMP int __cdecl     _cscanf (char*, ...);

_CRTIMP int __cdecl     _getch (void);
_CRTIMP int __cdecl     _getche (void);
_CRTIMP int __cdecl     _kbhit (void);
_CRTIMP int __cdecl     _putch (int);
_CRTIMP int __cdecl     _ungetch (int);


#ifndef     _NO_OLDNAMES

_CRTIMP int __cdecl     getch (void);
_CRTIMP int __cdecl     getche (void);
_CRTIMP int __cdecl     kbhit (void);
_CRTIMP int __cdecl     putch (int);
_CRTIMP int __cdecl     ungetch (int);

#endif      /* Not _NO_OLDNAMES */

#ifdef      __cplusplus
}
#endif

#endif      /* Not RC_INVOKED */

#endif      /* Not _CONIO_H_ */
```

5.4 #if, #elif, #else ir #endif

Prekompiliatorius turi sąlygines direktyvas, kuriomis galima keisti programos tekstą priklausomai nuo sąlygų.
Sintaksė:

```
#if konstantinė-sąlyga-1
sekcija-1   // naujoje-eilutėje
  ...
#elif konstantinė-sąlyga-2
sekcija-2   // naujoje-eilutėje
  ...
#elif konstantinė-sąlyga-n
sekcija-n   // naujoje-eilutėje
  ...
#else
galutinė sekcija
  ...
#endif
```

Pavyzdžiui, kai programa rašoma įvairioms platformoms ir yra skirtumai funkcijų pavadinimuose ar naudojime arba duomenų struktūrose ar dar kokie nors skirtumai, yra naudojama sąlyginė kompiliacija. Tai reiškia, kad yra naudojamos prekompiliatoriaus sąlyginės direktyvos.

5.5 #include

Direktyva **#include** įjungia į programos teksto kompiliavimą papildomus įrašus dažniausiai antraščių failus arba kitus (papildomus arba išorinius) programų failus.

Sintaksė:

```
#include <antraštės-failas>
```

arba

```
#include "source-file"
```

Kai antraštės vardas yra apgaubtas kampiniais skliausteliais <filename.ext>, antraštės failas yra ieškomas žinomuose antraščių kataloguose. Kelias iki šių katalogų paprastai būna įrašytas sistemos arba programavimo aplinkos kintamosiose. Kai antraštės vardas yra apgaubtas dvigubomis kabutėmis, visas tekstas yra naudojamas kaip kelias, jei nurodytas tik failo vardas, jis ieškomas tame pat kataloge, kur yra kompiliuojama programa. Visas antraštės failo tekstas, prekompiliacijos metu, yra įdedamas į programos tekstą toje vietoje, kur yra *#include* direktyva.

Pavyzdžiui:

```
#include <stdio.h>
#include "my_header.h"
#include "c:\work\project\my_lib.h"
```

5.6 #error

Direktyva **#error** nutraukia kompiliavimą ir grįžta su įrašytu klaidos pranešimu.

Sintaksė:

```
#error “Klaidos pranešimas”
```

Pavyzdžiui:

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

//#define VERSION "1.0"

#ifndef VERSION
#error "Nenustatyta programos versija"
#endif

int main(int argc, char *argv[])
{
    printf("VERSION %s\n",VERSION);
    system("Pause");
    return(0);
}
```

5.7 #pragma

Preprocesoriaus direktyva **#pragma**, arba dalis jos direktyvų yra palaikoma ne visuose kompiliatoriuose. Todėl reikia žiūrėti konkretaus kompiliatoriaus aprašymą. Jei direktyva nepalaikoma, ji yra ignoruojama.

Sintaksė:

```
#pragma direktyva
```

Pavyzdžiui:

```
#pragma argsused
```

direktyva veikia tik funkcijose ir ”išjungia“ pranešimą: "Parameter *name* is never used in function *func-name*"

6 Aprašytos makro komandos

Šios makro komandos yra aprašytos ir negali būti pakeistos.

__LINE__	Dešimtainis skaičius atitinkantis programos teksto eilutę.
__FILE__	Programos teksto failo vardas (eilutė).
__DATE__	Programos failo kompiliavimo data (eilutė). Formatas "mmm dd yyyy", toks pat kaip generuojamas funkcijos asctime().
__TIME__	Programos failo kompiliavimo laikas (eilutė). Formatas "hh:mm:ss", toks pat kaip funkcijos asctime().
__STDC__	Dešimtainė konstanta lygi 1. Naudojama parodyti, kad naudojamas standartinis C kalbos kompiliatorius.

Jos dažniausiai yra naudojamos programų versijų kontrolei, derinimo pranešimuose, derinant sudėtingas programas, programų serverių pranešimų failų įrašams formuoti.

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

int main(int argc, char *argv[])
{
    ;
    printf("Msg 101 from %s line %03d (build %s %s)\n",
                __FILE__, __LINE__, __DATE__, __TIME__);
    ;
    ;
    ;
    ;
    printf("Msg 102 from %s line %03d (build %s %s)\n",
                __FILE__, __LINE__, __DATE__, __TIME__);
    ;
    ;
    ;
    printf("Msg 103 from %s line %03d (build %s %s)\n",
                __FILE__, __LINE__, __DATE__, __TIME__);

    printf("\n");
    system("PAUSE");
    return 0;
}
```

```
Msg 101 from D:\C\Macros.c line 007 (build Dec 26 2007 20:03:34)
Msg 102 from D:\C\Macros.c line 012 (build Dec 26 2007 20:03:34)
Msg 103 from D:\C\Macros.c line 016 (build Dec 26 2007 20:03:34)

Press any key to continue . . .
```

Failo kompiliavimo metu makro komandos yra pakeičiamos programos vardu su visu keliu, eilutės numeriu, kurioje programos tekste yra makrokomanda __LINE__. Makro komandos __DATE__ ir __TIME__ yra pakeičiamos data ir laiku kada buvo kompiliuota programa.

7 Duomenų tipai

Jau kalbėjome apie tai, kad kompiuteriai dirba tik su skaičiais, taip pat ir raidės yra užkoduotos skaičiais. Iš matematikos prisimename, kad skaičiai yra sveikieji ir realieji. Realieji skaičiai yra trupmeniniai skaičiai. Paprastosios trupmenos, kaip pavyzdžiui viena trečioji (1/3) kompiuteriuose dažniausiai nenaudojamos. Kompiuteriuose dažniausiai naudojamos dešimtainės trupmenos realiesiems skaičiams atvaizduoti.

C kalboje, kaip ir daugelyje kitų programavimo kalbų, visi kintamieji turi turėti tipą. Tipas tai ne tik užimamos kompiuterio atminties dydis, bet ir skaičių ar simbolių atvaizdavimo galimybė, o taip pat skaičiavimo tikslumas. Taip kaip kintamieji, tipus turi turėti ir funkcijos, nes jos dažniausiai grąžina reikšmes jas iškvietusiai programai. Griežtas kintamųjų tipizavimas gali atrodyti nereikalingas ir net trukdantis, beje taip ir yra. Tačiau toks griežtumas labai padidina programų patikimumą, o taip pat padeda išvengti klaidų programuojant. Pradedantieji programuotojai labai greitai susiduria su tipizavo privalumais ir trūkumais. Viena labai dažnai programose naudojamų funkcijų yra *printf()*, kuri išspausdina duomenis ekrane. Deje šioje funkcijoje nėra galimybės nustatyti nei argumentų skaičiaus nei jų tipų, todėl šioje funkcijoje nei argumentų skaičius nei tipai nėra tikrinami. Todėl funkcija *printf()*, padarius klaidą aprašyme, gali ne tik neteisingai spausdinti, bet ir sugriauti visą programą, arba įvesti kompiuterio procesorių į begalinio ciklo vykdymą (kompiuterio pakibimas).

7.1 Kintamųjų deklaracija

Prieš naudojant kintamąjį ar funkciją jį pirmiausiai reikia deklaruoti, pranešti programai apie jo egzistavimą. Duomenų tipas yra suteikiamas deklaruojant kintamąjį ar funkciją. Kintamojo deklaracijos sintaksė yra:

duomenų-tipas *vardas;*
duomenų-tipas *vardas = reikšmė;*

Pavyzdžiui:

```
char        c;                                  // vienas baitas
int         i = 0;                              // keturi baitai
short int   x;                                  // du baitai
float       _pi = 3.141593;                     // keturi baitai
char        c_array[10]={0,3,9,1,7,5,2,6};      // dešimt baitų
int         i_array[10]={0,3,9,1,7,5,2,6};      // 40 baitų
```

Funkcijų deklaracija kiek sudėtingesnė, bet iš esmes tokia pat kaip ir kintamųjų. Funkcijos deklaracijos sintaksė:

duomenų-tipas *funkcijos_vardas(duomenų-tipas parametras1, ...);*

Duomenų tipas nurodytas prieš funkcijos vardą, nurodo kokio duomenų tipo atsakymą grąžins funkcija. Duomenų tipai, nurodyti už funkcijos vardo lenktiniuose skliausteliuose, nurodo kokio tipo parametrai turi būti perduodami funkcijai.

Pavyzdžiui *math.h* deklaruota kėlimo laipsniu funkcija *pow()* grąžina *double* tipo reikšmę kaip rezultatą, o kaip argumentai yra perduodamos dvi *double* tipo reikšmės:

```
_CRTIMP     double __cdecl pow (double, double);
```

7.2 Baziniai duomenų tipai

Skaičiavimams 8 bitų pakanka retai, nes aštuoni bitai yra skaičius nuo 0 iki 255 arba nuo -127 iki +127 arba ASCII simbolis. Todėl C kalboje yra daugiau duomenų tipų, skirtų įvairiems skaičiavimams ir loginiams veiksmams. Pagrindiniai C kalbos duomenų tipai pateikti 7.1 lentelėje.

Baziniai C kalbos duomenų tipai

7.1 lentelė.

Tipas	Dydis bitais	Ribos		Panaudojimas
unsigned char	8	0	255	Maži sveikieji teigiami skaičiai ir visi ASCII simboliai
Char	8	-127	127	Maži sveikieji teigiami ir neigiami skaičiai ir pagrindiniai ASCII simboliai
Enum	16	-32,768	32,768	Tvarkinga sveikųjų skaičių eilė
unsigned short int	16	0	65,535	Nedideli sveiki skaičiai ir teigiamų skaičių eilė
short int	16	-32,768	32,768	Nedideli sveikieji skaičiai su ženklu
unsigned int	32	0	4,294,967,295	Dideli sveikieji skaičiai ir teigiamų skaičių eilė
Int	32	-2,147,483,648	2,147,483,648	Dideli sveikieji skaičiai su ženklu
unsigned long	32	0	4,294,967,295	Labai dideli sveikieji skaičiai. Astronominiams atstumams
Long	32	-2,147,483,648	2,147,483,648	Labai dideli skaičiai su ženklu
Float	32	$3.4x10^{-38}$	$3.4x10^{38}$	Nedidelio tikslumo realieji skaičiai. Moksliniams skaičiavimams 7 skaitmenų tikslumu
double	64	$1.7x10^{-308}$	$1.7x10^{308}$	Didelio tikslumo realieji skaičiai. Moksliniams skaičiavimams 15 skaitmenų tikslumu
long double	80	$3.4x10^{-4932}$	$3.4x10^{4932}$	Labai didelio tikslumo realieji skaičiai. Finansiniams skaičiavimams 19 skaitmenų tikslumu

Duomenų tipus gali susikurti ir pats vartotojas. Tuo tikslu C kalboje yra numatyta priemonė – raktinis žodis **typedef** nurodantis, kad yra nustatomas naujas duomenų tipas. Skirtingi duomenų tipai yra reikalingi saugoti skirtingo dydžio skaičiams ir skirtingo tikslumo skaičiavimams.

Matome, kad skaičiavimams yra naudojami įvairūs duomenų tipai. Jie reikalingi, kad taupyti kompiuterio atmintį, be to didesnių ir tikslesnių skaičių skaičiavimui reikia ir daugiau procesoriaus laiko. Diuomenų tipo užimamas atminties dydis yra matuojamas baitais arba bitais. Baitą sudaro aštuoni bitai.

7.3 Duomenų tipas (bool)

Duomenų tipas **bool** turi dvi fiksuotas reikšmes **true** ir **false**. Tipas bool yra priskiriamas sveikųjų skaičių tipams. Reikšmę **false** atitinka skaičius **0**, o reikšmę **true** atitinka skaičius **1**.

```
bool  _found=false;
```

Pastaba: Ne visi C kalbos kompiliatoriai turi aprašytą *bool* tipą.

7.4 Duomenų tipas (void)

C kalboje yra dar vienas specialus duomenų tipas **void**. Šis tipas naudojamas:

- Pažymėti, kad funkcija neturi parametrų, funkcijos deklaracijoje:
```
int func(void);          // funkcija neturi parametrų
```
- Deklaruojama funkcija negrąžina jokios reikšmės:
```
void func(int a);        // funkcija negrąžina reikšmės
```
- Kaip bendra rodyklė (generic pointer) į bet kurį tipą:
```
void *ptr;               // rodyklė į iš anksto nežinomą tipą
```
- Tipų priskyrimui (typecasting):
```
extern int errfunc();    // klaidos kodas
 . . .
(void) errfunc();        // negrąžinti klaidos kodo
```

Programos pavyzdys, kuriame naudojamas *void* duomenų tipas.

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

void Print_Square(int Number);
void InitRandom (void);
int Random(void);

int main()
{
    Print_Square(10);
    InitRandom ();
    printf("Random numbers: %d %d %d\n",Random(),Random(),Random());
    printf("\n");
    system("Pause");
    return 0;
}

void Print_Square(int Number)
{
   printf("%d squared is %d\n",Number, Number*Number);
}

int Random(void)
{
    return( rand());
}

void InitRandom (void)
{
    srand((unsigned int)time((time_t *)NULL));
}
```

```
10 squared is 100
Random numbers: 30034 23701 22470

Press any key to continue . . .
```

7.5 Rodyklės (pointers)

Rodyklė yra specialus kintamojo tipas, kuri nurodo vietą arba adresą kito kintamojo arba tašką kur kintamasis prasideda. Kintamojo rodyklė sudaroma radus adresą kur šis kintamasis yra kompiuterio atmintyje. Tam C kalboje yra specialus operatorius **&**. Pavyzdžiui, jei turime *float* tipo kintamąjį *space*, nesunku surasti jo adresą ir jį priskirti rodyklei *space_ ptr*.

```
float space;
float * space_ptr, *address;

space_prt = &(space);
```

arba

```
address = &(space);
```

Apibendrinti operatorių & ir * funkcijas galima taip:

& kintamojo adresas (objekto vieta atmintyje)
* turinys to į ką rodo rodyklė (kintamojo nurodyto rodykle reikšmė)

Operatoriai **&** ir ***** visada rašomi prieš kintamąjį ir „priklijuojami prie jo“, kad nesumaišyti su bitine operacija ar daugyba.

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

int main()
{
    int  i;
    char *ptr;
    char *items[4][8]={"apple", "pear", "banana", "grape"};

    ptr = *items;

    for(i=0;i<4;i++) {
        printf ("\t%s\n", ptr);
        ptr = ptr+8;
    }
    printf("\n");
    system("Pause");
}
```

```
        apple
        pear
        banana
        grape

Press any key to continue . . .
```

Toliau pateikiama programa, kuri atspausdina visų kintamųjų adresus, reikšmes ir komentarus. Adresas – kintamojo vieta kompiuterio atmintyje.

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

int main(int argc, char *argv[])
{
  char name[9]={"Concorde"};
  short idx=0;
  double dval=3.14;
  char *nptr=&name[0];
  double *dvptr=&dval;
```

```
  printf(" Address   | Data      | Size | Comment\n");
  printf(" ---------|----------|------|--------\n");
  for(idx=7;idx>=0;idx--) {
    printf(" %08p | %8c | %4d | name[%d]\n",
      &name[idx],name[idx],sizeof(char),idx);
  }
  printf(" %08p | %8d | %4d | idx\n",&idx,idx,sizeof(short));
  printf(" %08p | %1.6f | %4d | dval\n",&dval,dval,sizeof(double));
  printf(" %08p | %08x | %4d | *nptr\n",&nptr,nptr,sizeof(size_t));
  printf(" %08p | %08x | %4d | *dvptr\n",&dvptr,dvptr,sizeof(size_t));
  printf("\n");

  system("PAUSE");
  return 0;
}
```

Programa ekrane išspausdina lentelę.

```
        Address   | Data      | Size | Comment
        ---------|----------|------|--------
        0022ff77 |         e |    1 | name[7]
        0022ff76 |         d |    1 | name[6]
        0022ff75 |         r |    1 | name[5]
        0022ff74 |         o |    1 | name[4]
        0022ff73 |         c |    1 | name[3]
        0022ff72 |         n |    1 | name[2]
        0022ff71 |         o |    1 | name[1]
        0022ff70 |         C |    1 | name[0]
        0022ff6e |        -1 |    2 | idx
        0022ff60 |  3.140000 |    8 | dval
        0022ff5c |  0022ff70 |    4 | *nptr
        0022ff58 |  0022ff60 |    4 | *dvptr

Press any key to continue . . .
```

PASTABA: Adresai gali būti kitokie ir tai priklauso nuo daugelio priežasčių. Kaip šiuo atveju yra užimama atmintis, detaliai yra pavaizduota priede C.

7.6 NULL

Nors iš tikrųjų tai yra skaičius „0“, tačiau NULL tipas yra neapibrėžtas (void), todėl ši reikšmė yra naudojama su įvairaus tipo duomenimis. Viena iš NULL reikšmių yra ta, kad rodyklė į kintamąjį ar objektą nenustatyta. Pavyzdžiui:

```
if ((fp=fopen("/etc/hosts","r") == NULL)
{
    exit(0);
}
```

reiškia, jei nepavyko atidaryti failo `"/etc/hosts"` programą užbaigti su signalu `exit(0)`.
Funkcija *fopen()*, kaip ir daugelis kitų C kalbos funkcijų, kurios grąžina rodykles, nesėkmės atveju grąžina NULL, o sėkmingai įvykdžius funkcija grąžina adresą į atidaryto failo valdymo bloką.

7.7 Konstantos (const)

Konstanta yra toks duomenų tipas, kuriam vieną kartą priskyrus reikšmę jos daugiau pakeisti nebegalima.

```
const double _e = 2.7182818282;
const double _pi = 3.14159265358979323846264338327950;
const double sqrt02 = 1.414213562373;
```

Pavyzdžiui bandymas vėliau programoje padaryti tokį priskyrimą

```
_e = 2.7182;
```

duos kompiliatoriaus klaidą.

7.8 Specialūs C kalbos simboliai

\b	ištrinti simbolį už kursoriaus ir nustatyti kursorių į jo vietą [backspace BS]
\f	naujas puslapis [form feed FF (also clear screen)]
\n	nauja eilutė [new line NL (like pressing return)]
\r	nustatyti kursorių į eilutės pradžią [carriage return CR (cursor to start of line)]
\t	horizontali tabuliacija, nutylint 8 pozicijos [horizontal tab HT]
\v	vertikali tabuliacija [vertical tab (not all versions)]
\"	dvigubos kabutės [double quotes (not all versions)]
\'	viengubos kabutės [single quote character ']
****	dešinysis įstrižas brūkšnys [backslash character \]
\ddd	aštuonetainis kodas gali būti ir ASCII
\xddd	šešioliktainis kodas gali būti ir ASCII

7.9 Vartotojo tipas (typedef)

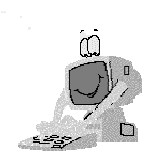
C kalba leidžia aprašyti savo sukurtą tipą, ar pervadinti jau esantį, naudojant *typedef* direktyvą. Šios direktyvos sintaksė:

typedef *duomenų-tipas naujas-tipas;*

Pavyzdžiui:

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

typedef     unsigned char     byte;
typedef     unsigned short    word;
typedef     unsigned int      dword;

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

    byte    name[30]={"Simple typedef test"};
    word    i, space, nospace;
    dword   length;

    nospace = 0;
    space = 0;
    for(i=0;i<30;i++) {
        if(name[i]==0) {
            length = i;
            break;
        }
        printf("%c",name[i]);
        if(name[i]==' ') {
            space++;
            continue;
        }
        nospace++;
    }
    printf("\n\n");
```

```
    printf("\t length      = %d\n", length);
    printf("\t exept space = %d\n", nospace);
    printf("\t space       = %d\n", space);

    printf("\n");
    system("Pause");
    return (0);
}
```



```
Simple typedef test

        length      = 19
        exept space = 17
        space       = 2

Press any key to continue . . .
```

7.10 Duomenų tipo pakeitimas (type cast)

Kaip buvo minėta, C kalboje visi kintamieji ir funkcijos yra tipizuoti t.y. visi kintamieji, konstantos ir funkcijos turi priklausyti kokiam nors duomenų tipui. Tačiau kartais reikia pakeisti kintamojo duomenų tipą, dažniausia tai pasitaiko, kai reikia pasinaudoti funkcija, kurios argumentai yra kitokio tipo negu turimas kintamasis.

Šio veiksmo sintaksė yra tokia:

(duomenų_tipas)kintamasis

Pavyzdžiui šioje programoje *div()* funkcijos argumentai gali būti *int* arba *long* tipo, tačiau reikia padalinti sveikus skaičius, kurie yra *double* tipo:

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

int main(int argc, char *argv[])
{
    div_t   res;
    double  d1=15, d2=4;

    res = div((long)d1, (long)d2);
    printf("div(%f, %f) = %d and %d/%d\n", d1, d2,
            res.quot, res.rem, (long)d2);

    printf("\n");
    system("Pause");
    return(0);
}
```

```
div(15.000000, 4.000000) = 3 and 3/4

Press any key to continue . . .
```

Duomenų tipo pakeitimas atliekamas prieš kintamąjį lenktiniuose skliausteliuose parašant reikiamo tipo vardą. C kalboje mažesnio tipo skaičiai yra pervedami į didesnio tipo skaičius, nenaudojant tipo keitimo.

Pavyzdžiui galima keisti char -> short int -> int -> long. Tačiau reikia nepamiršti apie skaičius su ženklu ir be ženklo. Kompiliatoriai dažniausiai tik įspėja, kad galima klaida, bet

programą sukompiliuoja. Tačiau, jei jusu mažesnio tipo skaičius yra su ženklu, o didesnio tipo skaičius yra be ženklo, galite pamesti neigiamus skaičius.

Kitokia, žymiai keblesnė, situacija, kai reikia didesnio tipo skaičius keisti mažesnio tipo skaičiais. Čia jau butina naudoti tipo pakeitimą, o už tai, kad programa veiks teisingai atsako programuotojas. Štai pavyzdys:

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

int main() {

    int i;

    for(i=-3; i<4; i++) {
        printf("value i=%4d of type (int) cast to (char) i=%4d\n",
                i, (char)i);
    }
    printf("\n");

    for(i=253; i<260; i++) {
        printf("value i=%4d of type (int) type cast to (char) i=%4d\n",
                i, (char)i);
    }
    printf("\n");

    for(i=-3; i<4; i++) {
        printf("value i=%4d of type (int) cast to (unsigned char) i=%4d\n",
                i, (unsigned char)i);
    }
    printf("\n");

    system("Pause");
    return (0);
}
```

```
Value i=  -3 of type (int) cast to (char) i=  -3
value i=  -2 of type (int) cast to (char) i=  -2
value i=  -1 of type (int) cast to (char) i=  -1
value i=   0 of type (int) cast to (char) i=   0
value i=   1 of type (int) cast to (char) i=   1
value i=   2 of type (int) cast to (char) i=   2
value i=   3 of type (int) cast to (char) i=   3

value i= 253 of type (int) type cast to (char) i=  -3
value i= 254 of type (int) type cast to (char) i=  -2
value i= 255 of type (int) type cast to (char) i=  -1
value i= 256 of type (int) type cast to (char) i=   0
value i= 257 of type (int) type cast to (char) i=   1
value i= 258 of type (int) type cast to (char) i=   2
value i= 259 of type (int) type cast to (char) i=   3

value i=  -3 of type (int) cast to (unsigned char) i= 253
value i=  -2 of type (int) cast to (unsigned char) i= 254
value i=  -1 of type (int) cast to (unsigned char) i= 255
value i=   0 of type (int) cast to (unsigned char) i=   0
value i=   1 of type (int) cast to (unsigned char) i=   1
value i=   2 of type (int) cast to (unsigned char) i=   2
value i=   3 of type (int) cast to (unsigned char) i=   3

Press any key to continue . . .
```

Pirmojoje grupėje atliekamas korektiškas tipo pakeitimas *int* tipo kintamasis su ženklu keičiami į *char* tipo kintamąjį.

Antroje grupėje matome, kaip *int* tipo kintamasis jį keičiant į *char* tipą, kurio reikšmė yra didesnė už 127 tampa neigiamu, o kai skaičius tampa 256 ir didėja pakeistas skaičius virsta 0 ir didėja.

Trečioje grupėje *int* tipo slaičiai keičiami nuo -3 iki 3, o pakeitus tipą į *unsigned char* nedideli neigiami skaičiai virsta dideliais skaičiais (253, 254, 255).

Iš šio pavyzdžio matosme, kad tipo pakeitimas turi būti atliekamas taip, kad neiškreiptų duomenų. Reikia tiksliai žinoti kokias reikšmes gali įgyti kintamasis, kurio tipas yra keičiamas.

Kitame pavyzdyje parodyta, kaip pakeičiant duomenų tipą, dalinant du sveikus skaičius, galima gauti skaičių su slankiu kableliu.

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

int main() {

    int     a=51, b=7;
    float   c;

    c=a/b;
    printf("%f=%d/%d\n\n", c, a, b);

    c=(float)a/b;
    printf("%f=%d/%d\n\n", c, a, b);

    c=a/(float)b;
    printf("%f=%d/%d\n\n", c, a, b);

    system("Pause");
    return (0);
}
```

```
7.000000=51/7

7.285714=51/7

7.285714=51/7

Press any key to continue . . .
```

Matome, kad pakeitus dalmens arba daliklio tipą į slankaus kablelio tipa (pavyzdyje *float*), rezultatas gaunamas skaičius su slankiu kableliu.

8 Duomenų struktūros

Baziniai duomenų tipai yra duomenų struktūros, įdiegtos kuriant programavimo kalbą. Kaip ir daugelyje programavimo kalbų, C kalboje yra šie baziniai duomenų tipai: simbolinis (char), sveikas (short, integer, long), realusis (float, double, long double), dvejetainis (bool), rodyklės tipas (pointer) ir du specialūs duomenų tipai tuščias (void) ir nulis (NULL).

Tačiau vien šių duomenų tipų kartais nepakanka. Taip pavyzdžiui teksto eilutė yra grupė simbolių ASCII simbolių, kur kiekvienas simbolis yra koduojamas vienu baitu. Kompleksinis skaičius sudarytas iš poros realiųjų skaičių, kurių vienas atitinka realiąją dalį, o kitas menamąją. Daugiakalbėse tekstų apdorojimo sistemose, daugelio kalbų palaikymui, vienas simbolis yra koduojamas dviem ar net keturiais baitais.

Matome, kad tokiais atvejais būtų patogu turėti kintamąjį, kurio viduje būtų keki kintamieji ir kuris iš išorės atrodytų kaip vienas kintamasis. Tam tikslui teikia kažkokių priemonių. Tokios priemonės ir yra duomenų struktūros.

8.1 Masyvai (Arrays)

C kalboje masyvus galima sudaryti iš bet kurio tipo kintamųjų. Masyve gali būti tik vieno tipo kintamieji. Masyvai gali būti vienmačiai ar daugiamačiai. Masyvo deklaracijos sintaksė yra tokia:

duomenų-tipas *vardas[elementai];*
duomenų-tipas *vardas[elementai] = {reikšmė1, ..., reikšmėN} ;*
duomenų-tipas *vardas[elementai_1] [elementai_2];*
duomenų-tipas *vardas[elementai_1] [elementai_2]... [elementai_N];*

```
int             i_array[5];             // vienmatis masyvas
double          d_array[4][4];          // dvimatis masyvas
unsigned char   img[640][480][3];       // trimatis masyvas
```

masyvai gali būti inicializuojami deklaruojant arba po deklaracijos. Kai inicializuojami keli masyvo elementai jie atskiriami kableliais ir apgaubiami figūriniais skliausteliais.

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

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

int i_array[5] = {0, 1, 2, 3 ,4};          // penkios reikšmes
char weak_day[7][3]={"Su","Mo","Tu","We","Th","Fr","Sa"};
double d_array[5][8];
int i, j;

    for(j=0; j<5; j++) {
        for(i=0; i<8; i++) {
            d_array[j][i] = (double)(j)+((double)(i)/10);
        }
    }

    printf("int i_array[5] = {0, 1, 2, 3 ,4};\n");
    for(i=0;i<5;i++)
        printf("%d",i_array[i]);
    printf("\n\n");

    printf("char weak_day[7][3]={\"Su\",\"Mo\",");
    printf("\"Tu\",\"We\",\"Th\",\"Fr\",\"Sa\"};\n");
    for(i=0;i<7;i++)
        printf("%s ",weak_day[i]);
    printf("\n");
    for(i=0;i<7;i++)
        printf("%c  ",weak_day[i][0]);
    printf("\n\n");

    printf("double d_array[5][8];\n");
    for(j=0; j<5; j++) {
        for(i=0; i<8; i++) {
            printf("%2.1f ",d_array[j][i]);
        }
        printf("\n");
    }
    printf("\n");

    system("PAUSE");
    return 0;
}
```

```
int i_array[5] = {0, 1, 2, 3 ,4};
01234

char
weak_day[7][3]={"Su","Mo","Tu","We","Th","Fr","Sa"};
Su Mo Tu We Th Fr Sa
S  M  T  W  T  F  S

double d_array[5][8];
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7
1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7
2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7
3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7
4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7

Press any key to continue . . .
```

Masyvai yra labai dažnai naudojami programose, kadangi masyvas yra sudarytas iš to paties tipo elementų ir yra labai patogu pasiekti bet kurį jo elementą. Kiekvienas masyvo elementas turi tam tikrą indeksą, kuris priklauso nuo vietos kurioje yra elementas. Masyvuose labai patogu saugoti kokius nors vienarūšius duomenis, pavyzdžiui grupės ar kurso visų egzaminų rezultatus, arba kokius nors statistinius rezultatus. Skirtingų rūšių elementus galima surašyti į skirtingus masyvus, o sąryšiui tarp skirtingų masyvų naudojami indeksai. Tuo pačiu indeksu skirtinguose masyvuose reikia saugoti vieno objekto duomenis. Panaudojant ciklus labai patogu apdoroti duomenis surašytus masyvuose.

Pirmojo masyvo elemento indeksas yra 0. Indeksas dažniausiai yra kintamasis, kurio tipas turi būti *int*.

Teksto apdorojimui, jis yra surašomas į simbolių (char tipo) masyvą. Simbolių masyvas, dar vadinamas eilute, o senesnėje literatūroje vadinamas literatu. C kalboje eilutė yra vienintelis būdas teksto apdorojimui. Deklaracijos metu labai paprasta inicializuoti simbolių eilutę (priskirti reikšmę), pakanka tekstą apgaubti dvigubomis kabutėmis ir figūriniais skliausteliais.

```
char name[9] = {"Concorde"};
```

Masyvo vardas yra rodyklė į masyvą, arba kitaip sakant visas masyvas pasiekiamas nurodant jo vardą. Pavyzdžiui printf funkcijoje, atspausdinti eilutei, reikia perduoti rodyklę į tekstą.

```
printf("Greičiausias civilinis lėktuvas yra %s\n", name);
```

Masyvo elementus galima pasiekti, pasinaudojant indeksu, skaičiumi įrašytu laužtiniuose skliausteliuose po masyvo vardo. Pirmojo masyvo elemento indeksas visada yra 0.

```
name[0] = C;
name[1] = o;
name[2] = n;
name[3] = c;
name[4] = o;
name[5] = r;
name[6] = d;
name[7] = e;
name[8] = '\0';   // eilutė visada turi baigtis 'null' simboliu
```

Pavyzdžiui eilutei „Concorde“, kurią sudaro 8 raidės, reikia masyvo iš 9 elementų, nes C kalboje eilutė turi baigtis simboliu „null“ (0) ir jam būtina numatyti vietą.

	n+0	n+1	n+2	N+3	n+4	n+5	n+6	n+7	n+8
char name[9]={“Concorde”}	C	o	n	c	o	r	d	e	‘\0’

Kiek masyvas užima vietos atmintyje galima suskaičiuoti pagal formulę ir pasinaudojus specialia funkcija *sizeoff(type)*:

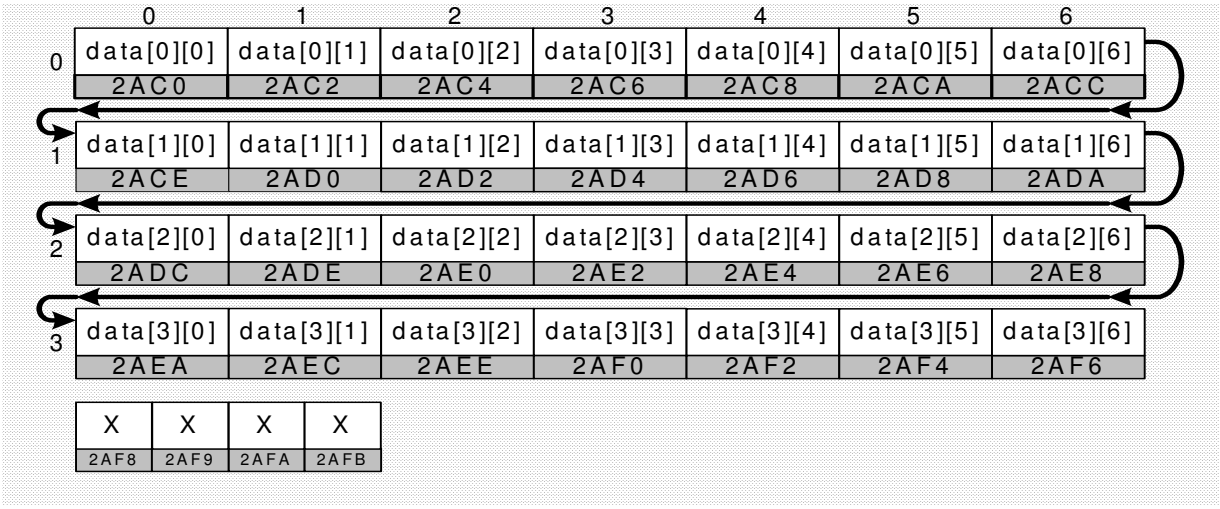
```
array_size = sizeof(type) * elements;
```

čia *sizeof(type)* – C kalbos funkcija grąžinanti tipo (*type*) dydį baitais, o minėtam masyvui *name[8]* *elements* = 8.

Masyvui, kompiuterio atmintyje, yra išskiriama tiek baitų, kiek baitų sudaro visas masyvas. Atmintis išskiriama vienoje vietoje, t.y. nenutrūkstamoje atminties dalyje. Pavyzdžiui turime dvimatį masyvą:

```
short int data[4][7];
```

Panagrinėkime, kaip išsiėsto masyvas kompiuterio atmintyje (3 pav.) Duomenų tipas *short int* yra 2 baitai. Todėl masyvą sudaro 2*4*7=56 baitai (0x38 šešioliktainėje sistemoje). Tarkime, atmintis buvo išskirta nuo 0x2AC0 adreso, pridėjus masyvo ilgį, randame adresą kur baigiasi masyvas
0x2AC0 + 0x38 = 0x2AF8. Nuo šio adreso prasideda kiti duomenys ar programos kodas.



8.1 pav. Masyvo atvaizdavimas kompiuterio atmintyje

8.2 Struktūros (Structures)

Duomenų struktūros gali apjungti kelių skirtingų tipų duomenis. Struktūros aprašymo sintaksėje yra naudojamas C kalbos raktinis žodis **struct**.

```
struct modelis {
        duomenų-tipas1      elementas1;
        duomenų-tipas2      elementas2;
        ....
        duomenų-tipasN      elementasN;
} objekto_vardas;
```

Pavyzdžiui tokia struktūra:

```
struct StrSoft {
      unsigned int    idx_m;
      unsigned long   idx_l;
      char            name[80];
      double          price;
```

```
        char            currency[5];
} s = {727,23, "Windows XP Home Edition", 335.18, "LT"},  *sptr =&s;
```

čia *StrSoft* yra modelio pavadinimas, o struktūrą atitinkančio objekto vardai yra *s* ir rodyklė **sptr*.

Struktūros dalyviai (elementai) pasiekiami naudojant išrinkimo operatorius (**.**) ir (**–>**). Rodyklė užrašoma iš simbolio minus (–) ir ženklo daugiau (>) be tarpelio (–>). Operatorius (**.**) yra vadinamas tiesioginio priėjimo (direct access) ir naudojamas pasiekti struktūros pažymėtos **s** dalyviams, o operatorius (**–>**) netiesioginio priėjimo (indirect access) ir naudojamas pasiekti tos pačios struktūros, tik pažymėtos kaip rodyklė ***sptr** dalyviams.

```
s.idx_m = 722;     // priskiriama reikšmė struktūros s dalyviui idx_m
sptr->price = 10; // priskiriama reikšmė struktūros s dalyviui price

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

struct StrSoft {  // deklaruojame struktūra kaip globalu kintamaji
      unsigned int    idx_m;
      unsigned long   idx_l;
      char            name[80];
      double          price;
      char            currency[5];
};

int main(int argc, char *argv[])
{
    struct StrSoft _soft[] = {
        {713,21, "Solaris", 10, "JPE"},
        {727,23, "Windows XP Home Edition", 335.18, "LT"},
        {727,24, "Windows XP Professional", 550.94, "LT"},
        {731,25, "Linux Slackware", 25, "USD"},
        {731,26, "SUSE Linux 10.0", 51.68, "EUR"},
        {727,27, "Office Professional 2003", 1163.10, "LT"},
        {0, 0, "", 0, ""}
                                };
    int i=0;

    printf("    ID    |              Name              |");
    printf(" Price     | Cur \n");
    printf(" --------|--------------------------------| ");
    printf("----------|-----\n");

    while(_soft[i].idx_m != 0 && _soft[i].idx_l != 0) {
        printf("%5ld-%-3ld| %-30s | %8.2f | %-3s\n",
                _soft[i].idx_m,_soft[i].idx_l,_soft[i].name,
                _soft[i].price,_soft[i].currency);
        i++;
    }

    printf("\n");
    system("PAUSE");
    return 0;
}
```

```
    ID    |              Name              | Price     | Cur
 --------|--------------------------------|-----------|-----
  713-21 | Solaris                        |     10.00 | JPE
  727-23 | Windows XP Home Edition        |    335.18 | LT
  727-24 | Windows XP Professional        |    550.94 | LT
  731-25 | Linux Slackware                |     25.00 | USD
  731-26 | SUSE Linux 10.0                |     51.68 | EUR
  727-27 | Office Professional 2003       |   1163.10 | LT
Press any key to continue . . .
```

Struktūros dalyviu gali būti bet kuris C kalbos duomenų tipas, taip pat ir kita, prieš tai deklaruota struktūra. Modelio pavadinimas yra naudojamas kaip duomenų tipas, kai reikia deklaruoti struktūrą pavyzdžiui kitoje struktūroje.

```
struct Persone {
      long  p_code;
      char  f_name[21];
      char  l_name[21];
} prsn;

struct supplier {
      long        index;
      Persone     prsn; // prieš tai aprašyta struktūra struct Persone
};
```

8.3 Junginiai (Unions)

Junginiai savo sintakse yra labai panašūs į struktūras. Pagrindinis skirtumas yra tas, kad tuo pačiu momentu gali būti „aktyvus“ tik vienas junginio dalyvis. Junginio dydis toks koks yra didžiausio junginio dalyvio dydis. Deklaruojant junginį yra naudojamas C kalbos raktinis žodis **union**.

union *modelis {*
duomenų-tipas1 elementas1;
duomenų-tipas2 elementas2;
....
duomenų-tipasN elementasN;
} objekto_vardas;

```
union myunion {
      int    i;
      double d;
      char   ch;
} mu, *muptr = &mu;
```

myunion tipo **mu** identifikatorius gali talpinti 2 baitus **int** kintamajam, 8 baitus **double**, arba 1 baitą **char** tipo kintamajam. Tačiau **myunion** kintamasis visada yra 8 baitų dydžio. Junginio dalyviai pasiekiami taip pat kaip struktūros dalyviai.

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

typedef union myunion {
      int         i;
      double      d;
      char        ch;
};

int main () {

    myunion  mu, *muptr = &mu;

    mu.ch = 'X';
    printf("mu.ch = 'X';\n");
    printf("\tmu.ch = %c \n", mu.ch); // mu.ch =  X
    printf("\tmu.i = %d \n", mu.i);   // neteisingas rezultatas
    printf("\tmu.d = %f \n", mu.d);   // neteisingas rezultatas

    muptr->i = 1234567;
    printf("muptr->i = 1234567;\n");
    printf("\tmuptr->ch = %c \n", muptr->ch); // neteisingas rezultatas
```

```
    printf("\tmuptr->i = %d \n", muptr->i);    // mu.i =  1234567
    printf("\tmuptr->d = %f \n", muptr->d);    // neteisingas rezultatas

    mu.d = 1.41421356;
    printf("mu.d = 1.41421356;\n");
    printf("\tmu.ch = %c \n", mu.ch); // neteisingas rezultatas
    printf("\tmu.i = %d \n", mu.i);    // neteisingas rezultatas
    printf("\tmu.d = %f \n", mu.d);    // mu.d =  1.41421356

    printf("\n");
    system("Pause");
      return (0);
}
```

```
        mu.ch = 'X';
                mu.ch = X
                mu.i = 88
                mu.d = 0.000000
        muptr->i = 1234567;
                muptr->ch = ç
                muptr->i = 1234567
                muptr->d = 0.000000
        mu.d = 1.41421356;
                mu.ch = ■
                mu.i = 1708926943
                mu.d = 1.414214
Press any key to continue . . .
```

8.4 Bitų laukas (bit field)

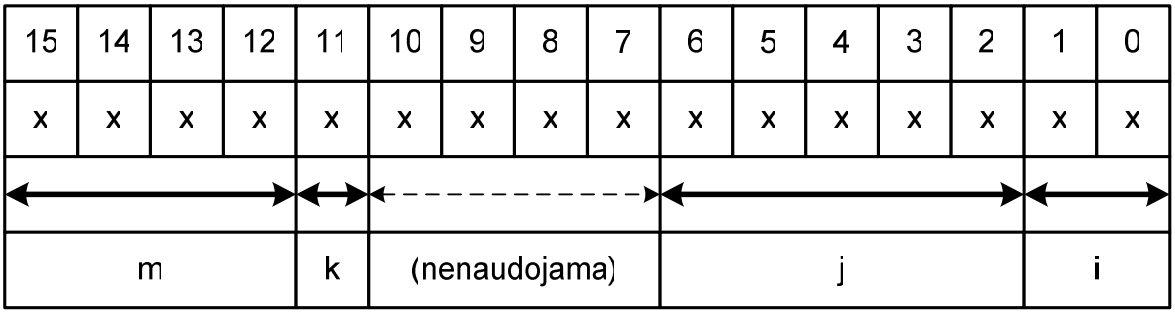
Kitas į struktūrą panašus tipas. Bitų laukas gali turėti kaip skaičius be ženklo, taip ir su ženklu. Bitų laukas aprašomas pagal tokį šabloną:

struct *modelis {*
duomenų-tipas1 elementas1: ilgis1;
duomenų-tipas2 elementas2: ilgis2;
...
duomenų-tipasN elementasN: ilgisN;
} objekto_vardas;

Pavyzdžiui:

```
struct my_bit_field {
      int          i : 2;
      unsigned int j : 5;
      int            : 4;
      int          k : 1;
      unsigned int m : 4;
} a, b, c;
```

Toks bitų laukas užima 2 baitus ir pavaizduotas 4 paveiksle.



8.2 pav. Bitų laukas

Bitų laukas patogus naudoti mikrovaldiklių ir mikroprocesorių būsenos registrų analizei ir valdymo registrų reikšmių nustatymui. O taip pat, kai reikia „suspausti“ didelį kiekį duomenų išlaikant tam tikrą struktūrą. Pavyzdžiui telefoninių pokalbių apskaitos duomenyse:– telefono numerį pakeitus iš simbolinio (20 ženklų) į skaičių pakanka 62 bitų, 6 skambučio rūšis galima užkoduoti 3 bitais, 4 bitais galima užkoduoti 16 mokėjimo planų, 3 bitais paros tarifus, šešiaženklį pokalbių laiką 18 baitų viso 90 bitų (12 baitų). Naudojant simbolinį formatą reikėtų 30 baitų, skaitiniam 15 baitų. Jei per vieną dieną įrašoma vienas milijonas įrašų lyginant su skaitiniu formatu yra sutaupoma trys megabaitai per dieną.
Bitų lauką patogu naudoti mikrovaldikliuose kuriuose valdymo registruose yra apjungtos kelios valdymo funkcijos ar darbo režimų nustatymai. Pavyzdžiui nuoseklios sąsajos valdymo registras.

RS-232 valdymo registras

8.1 lentelė

RS-232 Line Control Register

Bit 7	1	Divisor Latch Access Bit		
	0	Access to Receiver buffer, Transmitter buffer & Interrupt Enable Register		
Bit 6	Set Break Enable			
Bits 3, 4, 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0	One Stop Bit		
	1	2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.		
Bits 0, 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

RS-232 sąsajos valdymo registro aprašymas panaudojant bitų lauką galėtų būti toks:

```
#define WordLength_5    0
#define WordLength_6    1
#define WordLength_7    2
#define WordLength_8    3

#define StopBitLen_1    0
#define StopBitLen_2    1

#define NoParity        0
#define OddParity       1
#define EvenParity      3
#define HighParity      5
#define LowParity       7
```

```
#define SetBreakDisable 0
#define SetBreakEnable  1

#define Access          0
#define DivisorLatch    1

struct LineControlRegister {
      unsigned char WrdLen:2;
      unsigned char StopBitLen:1;
      unsigned char Parity:3;
      unsigned char SetBreak:1;
      unsigned char Access:1;
} LCR;


// set word length 8 bits, 1 stop bit, even parity,
// break enable, Access to Receiver buffer,
// Transmitter buffer & Interrupt Enable Register

LCR.Access=Access;                  // 0
LCR.SetBreak=SetBreakEnable;        // 0
LCR.Parity=EvenParity;              // 3
LCR.StopBitLen=StopBitLen_1;        // 0
LCR.WrdLen=WordLength_8;            // 3
```

Nuoseklios sąsajos RS-232 registro būsena

8.2 lentelė

Field Name	**Access**	**Break Enable**	**Parity Select**			**Stop Bit**	**Word Lenght**	
Bit Number	7	6	5	4	3	2	1	0
Bit Value	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
LCR	**0**	**0**	**3**			**0**	**3**	

8.6 Išvardijimo (enum) tipas

Išvardijimo tipas *enum* yra kilęs iš anglų kalbos trumpinio nuo *enumerated data*. Aukšto lygio kalbose yra naudojami žodžiai vietoje skaičių ir raidžių. Tam ir yra skirtas *enum* tipas. Jo sintaksė tokia:

```
enum modelis { vardas1, vardas2, ..., vardasN };
```

Pavyzdžiui Sulės sistemos planetų sąrašas:

```
enum planets {
            Mercury, Venus,   Earth,
            Mars, Jupiter, Saturn,
            Uranus, Neptune,Pluto
};
```

, čia Mercury = 0, Venus = 1, ..., Pluto = 8;

Matome, kad planetos yra sunumeruotos nuo nulio (0), tačiau numeraciją galima pakeisti. Dažnai tenka kažką daryti su datomis, o mėnesiai yra užkoduoti skaičiais. Tada galima sudaryti *enum* tipą mėnesiams.

```
enum month {
      January = 1, February, March, April, May, June, July,
      August, September, October, November, December };
```

, čia January = 1, February = 2, …, November = 11, December = 12.

Numeracija gali būti pakeista bet kurioje deklaracijos vietoje, už vardo parašius lygybę ir skaičių.

9 Kintamieji: lokalūs ir globalūs (variables)

Programavime kintamieji yra reikalingi duomenims saugoti, jie yra pažymimi vardais, kurie dar vadinami identifikatoriais.

C kalboje duomenų tipai yra naudojami kintamųjų deklaracijoje. Kintamojo deklaracija rezervuoja vietą kompiuterio atmintyje tiek baitų, kiek yra baitų kintamojo duomenų tipas. Kadangi kompiliatorius žino visų kintamųjų duomenų tipus, jis gali surasti visą eilę klaidų.

```
char    i;          // 1 baitas
char    name[20];   // 20 baitų
int     num;        // 2 baitai
long    index;      // 4 baitai
double  d_var[10];  // 80 baitų
```

Kintamieji, kurie yra deklaruoti funkcijoje, yra lokalūs kintamieji ir jie yra prieinami tik toje funkcijoje kurioje yra deklaruoti. Globalūs kintamieji turi būti deklaruoti prieš **main**() funkciją. Panagrinėkime žemiau pateiktą programą:

```
#include <stdio.h>
#include <stdlib.h>
int add_global (int i_val);    /* funkcijos add_global deklaracija */
int global_var;                // globalus kintamasis global_var

int main () {
    int     local_var, x_val;
      local_var = 6;
      global_var = 7;

      printf("<main 1> global_var = %d \t local_var = %d \n",
          global_var, local_var);
      x_val  = add_global (local_var);
      printf("<main 2> x_val = %d \n", x_val);
      /* Sekančioje eilutėje yra klaida, nes "loc_val" nedeklaruota */
      printf("<main 3> loc_val = %d \n", loc_val);
      printf("<main 4> global_var = %d \t local_var = %d \n",
          global_var, local_var);

    printf("\n");
    system("Pause");
      return (x_val);
}

int add_global (int i_val) {
      int loc_val;
      loc_val = i_val + global_var;

      printf("<add_global 1> global_var = %d \n", global_var);
/* Sekančioje eilutėje yra klaida, nes kintamasis "local_val"
   funkcijoje yra nematomas
*/
      printf("<add_global 2> local_var = %d \n", local_var);
      printf("<func 3> loc_val = %d \n", loc_val);
      global_var++;
      return (loc_val);
}
```

Po perprocesoriaus komandos #include seka funkcijos int add_global**(int, int)** deklaracija ir globalios kintamojo **int global_var** deklaracija. Reikia pastebėti, kad deklaruojant globalius kintamuosius jiems iš karto negalima priskirti reikšmių. Funkcijoje **main** deklaruojamas lokalus kinttamasis **local_var**. Abiem kintamiesiems priskiriamos reikšmės. Funkcija **printf**() galime išspausdinti abu kintamuosius, nes **global_var** yra pasiekiamas iš betkur, o **local_var** yra matomas **main** funkcijoje. **local_var** yra perduodamas **add_global**() funkcijai. Šioje funkcijoje **global_var** yra pasiekiamas, tačiau **local_var** yra nepasiekiamas, nes yra deklaruota **main** funkcijoje. Todėl **local_var** kintamojo reikšmę perduodame funkcijai **add_global**() kaip argumentą. Funkcija **add_global**() grąžina **local_var** ir **global_var** kintamųjų sumą, kuri priskiriama kintamajam **x_val**. Ši programa ekrane išspausdina:

```
<main 1> global_var = 7          local_var = 6
<add_global 1> global_var = 7
<func 3> loc_val = 13
<main 2> x_val = 13
<main 4> global_var = 8          local_var = 6
```

10 Operatoriai

Operatoriai yra simboliai, kurie nurodo, kokius veiksmus reikia atlikti su kintamaisiais, tarp kurių jie yra parašyti. C kalboje yra ne vien aritmetiniai operatoriai, bet ir loginiai, sąlyginiai, operacijų su bitais ir kiti. C++ kalbai realizuoti yra papildomi operatoriai objektų klasių ir klasių dalyvių priskyrimui.

Pagrindiniai C kalbos operatoriai

10.1 Lentelė.

Operatorius	Simbolinis žymėjimas	Funkcija
Sudėtis	+	Dvejetainė sudėtis (sudėtis)
Atimtis	-	Dvejetainė atimtia (atimtis)
Daugyba	*	Aritmetinė daugyba
Dalyba	/	Aritmetinė dalyba
Liekana	%	Aritmetinės dalybos liekana
Postūmis (shift)	<<	Į kairę
	>>	Į dešinę
Operacijos su bitais	&	AND (bitinė IR)
	^	XOR (bitinė išskirtinis ARBA)
	|	OR (bitinė ARBA)
Loginiai	&&	Loginė IR operacija
	||	Loginė ARBA operacija
Priskyrimo	=	Priskirti
	*=	Padauginti ir priskirti
	/=	Padalinti ir priskirti
	%=	Priskirti dalybos liekaną
	+=	Pridėti ir priskirti
	-=	Atimti ir priskirti
	<<=	Pastumti į kairę ir priskirti
	>>=	Pastumti į dešinę ir priskirti
	&=	Priskirti po loginės AND
	^=	Priskirti po loginės XOR
	|=	Priskirti po loginės OR
Priklausymo	<	Mažiau
	>	Daugiau
	<=	Mažiau arba lygu
	>=	Daugiau arba lygu
Sutapimo	==	Sutampa su
	!=	Nesutampa su
Komponentės išrinkimo	.	Tiesioginis išrinkimas
	–>	Kai turime rodyklė
Sąlygos	a ? x : y	Jei a tai x kitaip y
Didinimo (increment)	++	Kintamąjį padidina vienetu
Mažinimo (decrement)	--	Kintamąjį sumažina vienetu
Kablelis	,	Reiškinio dalių skirtukas
Klasės–dalyvių	::	Grupės aprašymas
	.	Nuoroda į klasės dalyvį
	–>	Nuoroda į klasės dalyvį

10.1 Aritmetinių veiksmų operatoriai

Veiksmų operatoriai C kalboje yra šie:

= + – * / % ++ – –

10.1.1 Priskyrimo operatorius =

Kitaip nei matematikoje, šis operatorius reiškia ne lygybę, bet priskyrimą, arba reikšmės kintamajam suteikimą. Kairėje lygybės pusėje esančiam kintamajam yra priskiriama dešinėje lygybės pusėje esančio reiškinio reikšmė (rezultatas) Pavyzdžiui:

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

int main(int argc, char *argv[])
{
int var00, var01 = 9, var02 = 5;
    var00 = var01 + var02 + 3;
    printf("\n\t%d = %d + %d + 3\n\n", var00,var01,var02);
    system("Pause");
    return(0);
}
```

```
        17 = 9 + 5 + 3

Press any key to continue . . .
```

po tokių veiksmų turime: *var01* reikšmė yra 9, *var02* reikšmė yra 5, o *var00* reikšmė yra 17.

C kalboje yra galima užrašyti ir tokius reiškinius:

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

int main(int argc, char *argv[])
{
int var00 = 0;
    printf("\n\t%d ... ", var00);
      var00 = var00 + 1;
    printf("%d ... ", var00);
      var00 += 1;
    printf("%d\n\n", var00);
    system("Pause");
    return(0);
}
```

```
        0 ... 1 ... 2

Press any key to continue . . .
```

čia yra paimama *var00* reikšmė ir prie jos pridedamas vienetas, o po to rezultatas priskiriamas *var00*.

10.1.2 Sudėties operatorius +

Sudeda abiejose sudėties ženklo pusėse esančius reiškinius

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

int main(int argc, char *argv[])
{
int var00, var01=7;

    var00 = var01 + 3;
    printf("\n\t%d = %d + 3\n\n", var00, var01);
    system("Pause");
    return(0);
}
```

Šioje programoje deklaruojame du **int** tipo kintamuosius var00 ir var01, kuriam iš karto priskirama reikšmė 7. Vėliau prie var01 yra pridedamas skaičius 3 ir rezultatas įrašomas į kintamąjį var00. Po to funkcija *printf()* atspausdinami visi kintamieji.

```
        10 = 7 + 3

Press any key to continue . . .
```

10.1.3 Atimties operatorius –

Iš reiškinio esančio kairėje operatoriaus „–“ pusėje atima reiškinį esantį dešinėje pusėje

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

int main(int argc, char *argv[])
{
int var00, var01=7;
      var00 = var01 - 3;
    printf("\n\t%d = %d - 3\n\n", var00, var01);
    system("Pause");
    return(0);
}
```

```
        4 = 7 - 3

Press any key to continue . . .
```

10.1.4 Ženklo pakeitimas –

Pakeičia reikšmės ženklą į priešingą

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

int main(int argc, char *argv[])
{
int var00 = 5, var01, var02;
```

```
      var01 = - var00;
      var02 = - var01;
    printf("\n\tvar00= %d var01= %d var02= %d\n\n",
            var00, var01, var02);
    system("Pause");
    return(0);
}
```

```
        var00= 5 var01= -5 var02= 5

Press any key to continue . . .
```

Įvykdžius programą turime *var00* reikšmė 5, *var01* reikšmė –5, *var02* reikšmė 5;

10.1.5 Daugybos operatorius *

Sudaugina reiškinius esančius abiejose operatoriaus „*“ pusėse

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

int main(int argc, char *argv[])
{
int var00, var01 = 5, var02;

      var02 = - var01;
      var00 = var01 * var02;
    printf("\n\t%d = %d * %d\n\n",
            var00, var01, var02);
    system("Pause");
    return(0);
}
```

```
    -25 = 5 * -5

Press any key to continue . . .
```

var00 reikšmė yra –25, *var01* reikšmė 5, *var02* reikšmė -5.

10.1.6 Dalybos operatorius /

Kairėje dalybos ženklo pusėje esantį reiškinį padalina iš dešinėje pusėje esančio reiškinio. Sveikųjų skaičių dalybos rezultatas visada yra sveikas skaičius. Jei dalmuo arba daliklis yra skaičius su slankiu kableliu, rezultatas bus skaičius su slankiu kableliu.

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

int main(int argc, char *argv[])
{
int var00, var01, var02;
      var01 = 8;
      var02 = 3;
```

```
    var00 = var01 / var02;
    printf("\n\t%d = %d / %d\n\n",
            var00, var01, var02);
    system("Pause");
    return(0);
}
```

```
        2 = 8 / 3

Press any key to continue . . .
```

var00 reikšmė yra 2.

10.1.7 Liekanos operatorius %

Grąžina sveikųjų skaičių dalybos liekaną.

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

int main(int argc, char *argv[])
{
int var00 = 11, var01 = 4, var02, var03;
    var02 = var00 / var01;
    printf("\nDalyba\t%d = %d / %d",var02, var00, var01);

    var03 = var00 % var01;
    printf("\nLiekana\t%d = %d %% %d\n\n",var03, var00, var01);

    system("Pause");
    return(0);
}
```

```
Dalyba  2 = 11 / 4
Liekana 3 = 11 % 4

Press any key to continue . . .
```

var02 reikšmė yra 3 ir *var03* reikšmė yra 1, nes 10/3 = 3 * 3 + 1

10.1.8 Operatoriai ++ (increment) ir – – (decrement)

Didinimo operatorius ++ (increment) kintamojo reikšmę padidina vienetu, o mažinimo operatorius – – (decrement) kintamojo reikšmę sumažina vienetu.

```
int var01 = 10;
int var02 = 10;
var01++;
--var02;
```

var01 reikšmė yra 11, *var02* reikšmė yra 9. Matome, kad šiuos operatorius galima rašyti ir iš kairės ir iš dešinės. Tai koks gi yra skirtumas? Skirtumas yra toks, kai operatorius (++) arba (--) yra parašytas prieš kintamąjį, kintamojo reikšmė pirmiausiai yra pakeičiama (padidinama arba sumažinama), o po to vykdomi reiškinio veiksmai, jei operatorius parašytas už kintamojo,

kintamojo reikšmė pakeičiama (padidinama arba sumažinama) tik tada, kai visi veiksmai reiškinyje būna atlikti.

```
int var00, var01, var02, var10, var11, var12;
var00 = 10;
var10 = 10;

var01 = ++var00;  // padidina ir priskiria
var02 = var00++;  // priskiria po to padidina
var11 = -- var10; // sumažina ir priskiria
var12 = var10- -; // priskiria po to sumažina
```

Čia pateikta programa demonstruoja increment ir decrement operatorių veikimą:

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

int main(int argc, char *argv[])
{
int var = 10;

    printf("var = %2d    var reiksme\n", var);
    printf("var = %2d    var reiksme veiksmo ++var metu \n", ++var);
    printf("var = %2d    var reiksme \n", var);
    printf("var = %2d    var reiksme veiksmo var++ metu \n", var++);
    printf("var = %2d    var reiksme \n", var);
    printf("var = %2d    var reiksme veiksmo --var metu \n", --var);
    printf("var = %2d    var reiksme \n", var);
    printf("var = %2d    var reiksme veiksmo var-- metu \n", var--);
    printf("var = %2d    var reiksme\n", var);

    printf("\n\n");
    system("Pause");
    return(0);
}
```

```
var = 10    var reiksme
var = 11    var reiksme veiksmo ++var metu
var = 11    var reiksme
var = 11    var reiksme veiksmo var++ metu
var = 12    var reiksme
var = 11    var reiksme veiksmo --var metu
var = 11    var reiksme
var = 11    var reiksme veiksmo var-- metu
var = 10    var reiksme


Press any key to continue . . .
```

10.1.9 Operacijos su konstantomis

Kai reikia pakeisti kintamojo reikšmę ir kitas kintamasis yra konstanta, veiksmų operacijas galima užrašyti trumpiau:

```
a += 5;    atitinka    a = a + 5;
a -= 5;    atitinka    a = a - 5;
a *= 5;    atitinka    a = a * 5;
a /= 5;    atitinka    a = a / 5;
a %= 5;    atitinka    a = a % 5;
```

10.2 Bitų operatoriai

C kalboje yra grupė operatorių darbui su bitais.

<<	Pastumti į kairę	<<=	Pastumti į kairę ir priskirti
>>	Pastumti į dešinę	>>=	Pastumti į dešinę ir priskirti
&	AND (bitinė IR)	&=	Priskirti po loginės AND
^	XOR (bitinė išskirtinis ARBA)	^=	Priskirti po loginės XOR
|	OR (bitinė ARBA)	|=	Priskirti po loginės OR
~	COMPLEMENT Bitinis vieneto papildymas		

10.2.1 Postūmio operacijos (shift)

Kad būtų paprasčiau nagrinėsime kaip vyksta postūmio operacijos baite (8 bitų grupėje). Kiekvienas bitas yra atvaizduojamas atskirame langelyje antroje eilutėje, o viršutinėje eilutėje yra surašytos bitų vertės (svorio koeficientai). Kodas bitais atitinka verčių ir atitinkamų bitų reikšmių sandaugų sumą. Pavyzdžiui dvejetainis kodas 00100110 atitinka skaičių

128x0+64x0+**32x1**+16x0+8x0+**4x1**+**2x1**+1x0 = 38

128	64	32	16	8	4	2	1
0	0	1	0	0	1	1	0

Į baitą *byte* įrašome 1 (00000001)

```
byte = 1;
```

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	1

atliekame postūmio į kairę vienu bitu operaciją

```
byte = byte << 1;
```

arba

```
byte <<= 1;
```

tada kintamajame *byte* bus reikšmė 2 (00000010)

128	64	32	16	8	4	2	1
0	0	0	0	0	0	1	0

Atliekame postūmį į kairę keturis bitus

```
byte = byte << 4;
```

arba

```
byte <<= 4;
```

128	64	32	16	8	4	2	1
0	0	1	0	0	0	0	0

dabar kintamajame *byte* yra reikšmė 32 (00100000). Apibendrinat galima pastebėti, kad operacija a **<<=** n; atitinka sandaugai $a \cdot 2^n$.

Dabar esamai *byte* reikšmei atlikime postūmį į dešinę

```
byte = byte >> 5;
```

arba

```
byte >>= 5;
```

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	1

Matome, kad vienetas pasistūmė penkiomis pozicijomis į dešinę. Arba kitaip tariant reikšmė buvo padalinta iš 2^5=32.

O kas bus jei yra ne vienas vienetas. Jei pavyzdžiui skaičių 7 ir pastumsime 3 pozicijas į kairę,

128	64	32	16	8	4	2	1
0	0	0	0	0	1	1	1

128	64	32	16	8	4	2	1
0	0	1	1	1	0	0	0

gauname $7 \bullet 2^3 = 7 \bullet 8 = 56$, o jei pastumsime dar 3 pozicijas į kairę

```
byte <<= 3;
```

	128	64	32	16	8	4	2	1
1	1	1	0	0	0	0	0	0

gauname 192 ir pastebime, kad dingo vienas vienetukas. Taip atsitiko dėl perpildymo, nes mes atliekame operacijas su vienu baitu.

Tokia pat situacija gaunasi ir stumiant į dešinę

128	64	32	16	8	4	2	1
0	0	1	1	1	0	1	1

```
byte >>= 4;
```

128	64	32	16	8	4	2	1				
0	0	0	0	0	0	1	1	1	0	1	1

10.2.2 Bitų loginės operacijos (AND, OR, XOR, COMPLEMENT).

AND	&	a = a & m	a &= m
OR	|	a = a | m	a |= m
XOR	^	a = a ^ m	a ^= m
NOT	~	a = ~a	a = ~a

Bitų loginės operacijos kartais dar vadinamos operacijomis su bitų kaukėmis (bit mask). Panagrinėsime bitų daugybos operaciją *&* (*AND*).

	0	0	1	1	1	0	1	1
&								
	0	0	0	0	1	1	1	1
=								
	0	0	0	0	1	0	1	1

Loginės daugybos operacija yra atliekama su kiekviena bitų pora. Bitų seka *00001111* dar vadinama kauke, nes, šiuo atveju, išskiria keturis jauniausius bitus.

Operacija ARBA (*OR*)

0	0	1	1	1	0	1	1
|							
1	1	1	1	0	0	0	0
=							
1	1	1	1	1	0	1	1

Matome, kad su *OR* operatoriumi galima nustatyti vieneto reikšmes norimuose bituose.

Operacija „IŠSKIRTINIS ARBA“ (*XOR*).

0	0	1	1	1	0	1	1
^							
1	1	1	1	0	0	0	0
=							
1	1	0	0	1	0	1	1

Operacija bitinis vieneto papildymas (*COMPLEMENT*).

0	0	1	1	1	0	1	1
~							
1	1	0	0	0	1	0	0

Ši operacija vadinama bitiniu vieneto papildymu arba *complement* todėl, kad sudėjus (atlikus bitų sudėties operaciją) prieš tai buvusį skaičių ir gautą skaičių visi bitai yra vienetai:

0	0	1	1	1	0	1	1
|							
1	1	0	0	0	1	0	0
=							
1	1	1	1	1	1	1	1

10.3 Loginiai operatoriai

Loginiai operatoriai reikalingi, kai programoje, esant skirtingoms sąlygoms, kurios yra aprašomos kintamųjų reikšmėmis, reikia atlikti skirtingus veiksmus. Pavyzdžiui visi žinome, kad dalyba iš nulio negalima. Bet programose gali pasitaikyti tokių situacijų, kuriose daliklis pasidaro lygus nuliui. Tada kompiuteryje kyla perpildymo klaidą. Taigi tokiai situacijai išvengti, reikia patikrinti ar daliklis yra lygus nuliui. Tokiems ir panašiems atvejams yra naudojami loginiai operatoriai.

C kalboje loginiuose operatoriuose yra naudojami šie raktiniai žodžiai: **if**, **else**, **switch**, **case**, **break**, **default.**

Loginiuose operatoriuose, sąlygų patikrinimui naudojamos sąlygų operacijos, kurių rezultatas yra loginis teigimas (true) arba loginis neigimas (false).

10.3.1 Sąlygų operacijos yra naudojamos loginiuose operatoriuose. Jos yra tokios:

< mažiau
<= mažiau arba lygu (nedaugiau)

==	lygu
!=	nelygu
>=	daugiau arba lygu (nemažiau)
>	daugiau

Matome, kad jos yra tokios pat kaip matematikoje. Tačiau C kalboje intervalams užrašyti negalima naudoti tokios pat konstrukcijos kaip matematikoje pavyzdžiui intervalą `-10 < X <10` C kalboje užrašome taip:

```
((X > -10) && (X < 10))
```

Sąlyga yra padalinama į dvi dalis ir jos sujungiamos loginės aritmetikos operatoriais.

10.3.2 Loginės aritmetikos operatoriai yra:

&&	IR	Loginė daugyba
||	ARBA	Loginė sudėtis
!	NE	Neigimas

Loginė aritmetika yra skirta skaičiavimams dvejetainėje skaičiavimo sistemoje t.y. sistemoje, kurioje yra tik du skaičiai **1** ir **0**, arba būsenos **Taip** ir **Ne.** Pavyzdžiui elektros jungiklis gali būti įjungtas arba išjungtas. Prisiminkime (jei dar nežinote tai susitarkime), kad **1** žymime **Teisingą** teiginį, o **0** žymime **Neteisingą** teiginį. Tada galima sudaryti tokias lenteles, kai **A** ir **B** keičiasi ir įgyja visas galimas kombinacijas, o **C** yra rezultatas.

IR

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

ARBA

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

NE

A	C
0	1
1	0

Angliškai šie operatoriai vadinami: **AND** operatorius IR, **OR** – ARBA, **NOT** – NE.

Panagrinėkime pavyzdžius, kuriuose yra naudojami loginiai, sąlygos ir loginės aritmetikos operatoriai:

1. Reikia patikrinti ar kintamajame *c* esantis ASCII simbolis yra skaičius. Pažiūrėję į ASCII lentelę matome, kad skaičių kodai prasideda nuo 48 (dešimtainis) simbolio **0** kodas ir baigiasi 57 simbolio **9** kodas. Vadinasi visi kodai iš intervalo nuo 48 iki 57 yra skaičių. Tada galime užrašyti tokią sąlygą:

```
(( c >= 48 ) && (c <= 57))
```

	46 .	47 /	48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	59 ;
c>=48	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
c<=57	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0
(c>=48)&&(c<=57	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0

2. Reikia patikrinti ar kintamajame *c* esantis ASCII simbolis yra bent vienas iš aritmetinio veiksmo simbolių t.y. */+–. Surandame kodus ir parašome sąlygą:

```
(c ==42 ||  c == 43 ||  c == 45  ||  c == 47)
```

	40 (	41)	42 *	43 +	44 ,	45 –	46 .	47 /	48 0	49 1
c==42	0	0	**1**	0	0	0	0	0	0	0
c==43	0	0	0	**1**	0	0	0	0	0	0
c==45	0	0	0	0	0	**1**	0	0	0	0
c==47	0	0	0	0	0	0	0	**1**	0	0
(c==42 || c==43 || c==45 || c==47)	0	0	**1**	**1**	0	**1**	0	**1**	0	0

Šią sąlygą galima užrašyti ir kitaip

```
( c ==42 ||  c == 43 ||  c == 45  ||  c == 47)
```

arba

```
((( c >41) && ( c < 44)) || ( c == 45 ) || ( c == 47))
```

Matome, kad sąlygos operatoriuose yra daug skliaustelių, čia yra parašyti ir nebūtini skliausteliai, tačiau kartais geriau daugiau skliaustelių nes dėl to lengviau suprasti programą, o kompiliatorius nebūtinus skliaustelius ignoruoja. Pavyzdžiui paskutinę sąlygą galima užrašyti taip:

```
((c >41 && c < 44) || c == 45 || c == 47)
```

nuo to veikimas visiškai nepasikeičia, o kaip aiškiau rinktis jums.

3. Reikia leisti įvedinėti simbolius iš klaviatūros kol bus paspaustas „Esc“ klavišas. Sąlygą galima suformuluoti ir taip: – kol ne „Esc“ skaityti klaviatūrą. Na o programos pavyzdys gali būti toks:

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

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

    char c;

    printf ("\'Esc\' baigti programai\n");
    while ((c=getch()) != 0x1b) {    // kol bus paspaustas 'Esc'
         printf("%c",c);              // spausdinti simbolá
    }
    printf("\n");
    system("Pause");
    return(0);
}
```

```
'Esc' baigti programai
qwerty 123456789 +-*/
Press any key to continue . . .
```

Sąlyga *(c != 0x1b)* yra teisinga visiems simboliams išskyrus „Esc“, kurio ASCII šešioliktainis kodas yra *1b*.

11 Programos vykdymo valdymo operatoriai

11.1 Operatoriai if ... else

Tai sąlygos operatorius, kuris reiškia – jeisąlyga teisinga (*if()*) vykdyti pirmąją veiksmų grupę – priešingu atveju (else) vykdyti antrąją veiksmų grupę.Loginis operatorius *if* gali būti naudojamas vienas, kai tuo tarpu *else* tik poroje su *if*.

Sintaksė:

```
if(sąlyga)
      veiksmas1                // kai sąlyga tenkinama
```

arba

```
if(sąlyga) {
      veiksmas1                // kai sąlyga tenkinama
      ...
      veiksmasN
}
```

arba

```
if(sąlyga)
      veiksmas1                // kai sąlyga tenkinama
else
      veiksmas2                // kai sąlyga netenkinama
```

arba

```
if(sąlyga) {
      veiksmas_TAIP_1    // kai sąlyga tenkinama
      ...
      veiksmas_TAIP_N
}
else {
      veiksmas_NE_1            // kai sąlyga netenkinama
      ...
      veiksmas_NE_N
}
```

Pavyzdžiui tokia programa:

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

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

    char c;

    printf("Baigti \'Esc\'\n");
    while ((c=getch()) != 0x1b) {        // kol c != "Esc"
        if(c==0x3E)                      // klavišas <
            printf("Daugiau %c\n", c);
        else if(c==0x3D)                 // klavišas =
            printf("Lygu %c\n", c);
        else if(c==0x3C)                 // klavišas >
            printf("Maziau %c\n", c);
        else
            printf("Kitas klavisas \'%c\' %d %02x\n",
                 c,(unsigned char)c, c);
    }
```

```
    printf("\n");
    system("Pause");
    return(0);
}
```

```
Baigti 'Esc'
Lygu =
Kitas klavisas 1 49 31
Maziau <
Daugiau >
Press any key to continue . . .
```

Panagrinėkime nedidelį programos fragmentą

```
if(c==0x3E)              // simbolis ">" (daugiau)
    printf("Daugiau %c\n", c);
else if(c==0x3C)         // simbolis "<" (mažiau)
```

pirmasis *if* operatorius patikrina sąlygą, ar kintamasis *c* yra lygus šešioliktainiam skaičiui *3E*. Pažiūrėję į ASCII kodų lentelę pamatysite, kad šešioliktainį skaičių 3E atitinka simbolis „>“. Tada, jei kintamajame *c* yra skaičius 0x3E, vykdoma eilutė *printf("Daugiau %c\n", c);*, kuri ekrane atspausdins *Daugiau >*. Tačiau, jei kintamajame *c* yra kitoks skaičius (ne 0x3E), programa „pereis“ prie eilutės *else*, kurios veiksmas yra kitas sąlygos operatorius, kuris analogiškai anksčiau išnagrinėtam, patikrina ar kintamajame yra simbolis „<“ (mažiau).

Šiame programos pavyzdyje matome ir kitokią sąlygos operaciją `((c=getch()) != 0x1b)`, ši sąlyga reiškia „nelygu Esc“, nes Esc šešioliktainis kodas yra 0x1b.

11.2 Sąlygos operatorius ? ... :

Sąlyginė operacija yra trumpas *if() .... else ....* užrašymas. Pavyzdžiui skaičiaus moduliui arba absoliutinei reikšmei gauti naudojama tokia išraiška:

```
y = (x > 0)? x : -y;
```

o su *if() .... else ....* užrašyti reikia taip:

```
if(x > 0)
    y = x;
else
    y = -x;
```

Arba galima užrašyti ir vienoje eilutėje:

```
if(x > 0) y = x; else y = -x;
```

11.3 Operatoriai switch, case, break, default

Šie operatoriai dar kartais vadinami perėjimo pagal lentelę operatoriais, iš analogijos su asemblerio kalba. Jų veikimas yra labai efektyvus, nes po switch() operatoriaus programos valdymas iškarto yra perduodamas į tą eilutę kurioje yra atitinkama case X: reikšmė.

Sintaksė:

```
switch (kintamasis) {
```

```
        case reikšmė1 : {
            veiksmas1 // kai kintamasis == reikšmė1
            break;
        {
        case reikšmė2 : {
            veiksmas2 // kai kintamasis == reikšmė2
            break;
        {
        default {
            veiksmas  // kai kintamasis neatitiko nei vienai reikšmei
        }
    }
```

Matome, kad yra sudaroma kintamojo galimų reikšmių lentelė ir aprašomi veiksmai kiekvienai reikšmei. Operatoriaus *default veiksmas* yra vykdomas kai kintamojo reikšmė neatitinka nei vienai aprašytai reikšmei. Panagrinėkime žemiau pateiktą programą.

```
/* ------------------------------------------------------------
                        switch.c
   ------------------------------------------------------------ */
   #include <conio.h>
   #include <stdio.h>
   #include <stdlib.h>

   int main(int argc, char* argv[])
   {
     int  c, c_int, sum=0;

     printf("\tIseiti is ciklo \'Esc\'\n\n");
     printf("\tIvedinekite skaicius, o programa skaiciuos ju suma:\n");
     while ((c=getch()) != 0x1b) {
       c_int = c - 48;   // atimame nulio koda ir gauname skaitmena
       switch (c) {
         case 48:        // simbolis 0
           sum += c_int;
           printf("Pridejus \'nuli\' suma nesikeicia   %5d\r", sum);
           break;
         case 49:        // simbolis 1
           sum += c_int;
           printf("Pridejus \'viena\' suma             %5d\r", sum);
           break;
         case 50:        // simbolia 2
           sum += c_int;
           printf("Pridejus \'du\' suma                %5d\r", sum);
           break;
         case 51:        // simbolis 3
           sum += c_int;
           printf("Pridejus \'tris\' suma              %5d\r", sum);
           break;
         case 52:        // simbolis 4
           sum += c_int;
           printf("Pridejus \'keturis\' suma           %5d\r", sum);
           break;
         case 53:        // simbolis 5
           sum += c_int;
           printf("Pridejus \'penkis\' suma            %5d\r", sum);
           break;
         case 54:        // simbolis 6
           sum += c_int;
           printf("Pridejus \'sesis\' suma             %5d\r", sum);
           break;
         case 55:        // simbolis 7
           sum += c_int;
           printf("Pridejus \'septynis\' suma          %5d\r", sum);
           break;
```

```
            case 56:        // simbolis 8
              sum += c_int;
              printf("Pridejus \'astuonis\' suma          %5d\r", sum);
              break;
            case 57:        // simbolis 9
              sum += c_int;
              printf("Pridejus \'devynis\' suma           %5d\r", sum);
              break;
            default:
              printf("O cia visai ne skaicius!          \r");
              break;
          }       // switch operatoriaus pabaiga
        }
        printf("\n");
        system("Pause");
        return 0;
}
```

```
        Iseiti is ciklo 'Esc'

        Ivedinekite skaicius, o programa skaiciuos ju suma:
Pridejus 'du' suma                    59
Press any key to continue . . .
```

Pirmiausia matome vieną nepažįstamą operatorių *while()*, tai ciklo operatorius apie kurį plačiau pakalbėsime truputį vėliau. Toliau yra programos konstrukcija su *switch()* operatoriumi, kuriam yra perduodamas iš klaviatūros nuskaityto simbolio kodas. Vėliau seka *case* operatorius, prie kurio parašytas skaičiaus – simbolio dešimtainis kodas iš ASCII lentelės. Atitikimam kodui *c* atliekamas sumavimas ir į ekraną yra spausdinama atitinkama eilutė *„Pridėjus ...“*. Operatorius *break* neturi jokių argumentų ir reiškia „nutraukti vykdymą“ tai yra nurodo, kad reikia išeiti iš *switch* operatoriaus.

12 Ciklai

Dažnai programoje reikia valdyti besikartojantį procesą. Sprendimas yra ciklai. Ciklai leidžia sudaryti veiksmų seką, kuri gali kartotis ir kartotis, su sąlygų mechanizmu, kuris leidžia užbaigti ciklo vykdymą. C kalboje yra trijų rūšių ciklai:

- while
- do ... while
- for

12.1 while ()

Ciklas *while* yra paprasčiausias iš visų ciklų. Šio ciklo sintaksė yra tokia:

```
while (sąlyga)
{
      Ciklo_veiksmas;
}
sekantis_veiksmas;
```

Pirmasis svarbus dalykas yra, kad ciklo sąlyga (pavyzdžiui a > b) yra apskaičiuojama kiekvieną kartą, kai pradedamas vykdyti ciklas. Ciklo *veiksmas* yra vykdomas, kol sąlyga yra teisinga. Kai sąlyga pasidaro neteisinga, arba reiškinys lenktiniuose skliausteliuose už žodžio *while* tampa lygus nuliui (0) ciklo vykdymas nutraukiamas ir vykdoma sekanti programos eilutė po ciklo *veiksmą* apgaubiančių figūrinių skliaustelių *sekantis_veiksmas*.

Antras svarbus dalykas, kurį reikia žinoti, yra tai, kad sąlyga yra tikrinama ne tik kiekvienos ciklo iteracijos pradžioje, bet ir pirmos iteracijos pradžioje. Tai yra svarbu, nes jei sąlyga yra **Neteisinga (0)** dar prieš prasidedant ciklui (prieš *while* operatorių), tai ciklo veiksmas nebus vykdomas nei karto.

Trečias svarbus dalykas – ciklo sąlygoje turi būti kintamasis, kuris keičiasi ciklo veiksme, arba nuo išorinio poveikio. Priešingu atveju ciklas bus begalinis t.y. niekada nesibaigs.

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

int main(int argc, char* argv[])
{
char    c;
int     i=0;

    printf("Vienoje eiluteje spausdinami 30 simboliu\n");
    printf("Isejimui is ciklo paspauskite 'Esc'\n\n");
    while ((c=getch()) != 0x1b) {   // kol c != "Esc"
        printf("%c", c);
        i++;
        // vienoje eilutėje spausdinti 30 simboli?
        if(i>=30) {
            i=0;
            printf("\n");
        }
    }
    printf("\n");
    system("Pause");
    return 0;
}
```

```
Vienoje eiluteje spausdinami 30 simboliu
Isejimui is ciklo paspauskite 'Esc'

123456789012345678901234567890
12345

Press any key to continue . . .
```

arba

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

int main(int argc, char* argv[])
{
    // ciklo kintamasis turi buti nustatytas prieš prasidedant ciklui
    int i=0;
    while(i<10) {              // kol ciklo kintamasis i<10
        printf("%2d ", i);     // spausdinamas dvieju simboliu skaicius
        i++;                   // ciklo kintamaji padidinam vienetu
    }
    printf("\n");
    system("Pause");
    return 0;
}
```

```
   0  1  2  3  4  5  6  7  8  9
Press any key to continue . . .
```

12.2 do .... while()

Šį ciklą galima apibūdinti, kaip *atlikti veiksmą kol sąlyga teisinga*.

```
do {
      veiksmas;
} while (sąlyga);
```

Reikia atkreipti dėmesį, kad šiame cikle sąlyga yra tikrinama ciklo pabaigoje. Taigi ciklas bus visada vykdomas bent vieną kartą, net kai ciklo pradžioje sąlyga yra **Neteisinga (0)**. Tai jo vienintelis skirtumas nuo *while()* ciklo.

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

int main(int argc, char* argv[]) {
    double  sin_val, itr, x1, x2, vpi, vmax;
    int     i, di;
    vpi =   3.1415926;          // PI
    itr =   0.16;               // Pokytis
    vmax = 2*vpi;
    di = (int)(vmax/(2*itr));
    i = 0;                      // Eiles numeris

    printf(" Nr\tx\tsin(x)\t Nr\tx\tsin(x)\n");
    printf("======================\t======================\n");
    do {                        // Ciklo pradzia
        x1 = itr*i;             // Apskaiciuojamas kampas
        sin_val = sin(x1);      // Kampo sinusas
        printf("%3d\t%2.3f\t%+2.3f\t", i, x1, sin_val);
        if(x1 <= vmax) {
            x2 = itr*(i+di);    // Apskaiciuojamas kampas
            printf("%3d\t%2.3f\t%+2.3f", i+di, x2, sin(x2));
        }
        printf("\n");
        i++;
    } while ((x1 <= vmax)&&(x2 <= vmax));

    printf("\n");
    system("Pause");
    return 0;
}
```

```
 Nr      x        sin(x)    Nr      x        sin(x)
=====================    =====================
  0     0.000    +0.000    19     3.040    +0.101
  1     0.160    +0.159    20     3.200    -0.058
  2     0.320    +0.315    21     3.360    -0.217
  3     0.480    +0.462    22     3.520    -0.369
  4     0.640    +0.597    23     3.680    -0.513
  5     0.800    +0.717    24     3.840    -0.643
  6     0.960    +0.819    25     4.000    -0.757
  7     1.120    +0.900    26     4.160    -0.851
  8     1.280    +0.958    27     4.320    -0.924
  9     1.440    +0.991    28     4.480    -0.973
 10     1.600    +1.000    29     4.640    -0.997
 11     1.760    +0.982    30     4.800    -0.996
 12     1.920    +0.940    31     4.960    -0.970
 13     2.080    +0.873    32     5.120    -0.918
 14     2.240    +0.784    33     5.280    -0.843
 15     2.400    +0.675    34     5.440    -0.747
 16     2.560    +0.549    35     5.600    -0.631
 17     2.720    +0.409    36     5.760    -0.500
 18     2.880    +0.259    37     5.920    -0.355
 19     3.040    +0.101    38     6.080    -0.202
 20     3.200    -0.058    39     6.240    -0.043
 21     3.360    -0.217    40     6.400    +0.117

Press any key to continue . . .
```

12.3 for (...;...;...)

Šis ciklo operatorius yra sudėtingiausias iš visų ciklo operatorių. Ciklas *for* visada turi kintamąjį, kuriuo manipuliuoja kiekvienoje ciklo iteracijoje. Jis yra vadinama *ciklo kintamuoju*. Ciklo operatoriaus *for* sintaksė yra tokia:

```
for (reiškinys1; sąlyga; reiškinys2) {
      Veiksmas;
}
```

Paprastai naudojant *for* operatorių, parametrai turi tokias reikšmes:

- *reiškinys1* paprastai čia yra inicializuojamas ciklo kintamasis t.y. nustatoma jo reikšmė ciklo pradžiai pavyzdžiui (*i = 0*).
- *sąlyga* tai sąlyga panaši į *while* ciklo, ji yra apskaičiuojama kiekvienai ciklo iteracijai ir kai sąlyga tampa **Neteisinga (0)** ciklas baigiamas. Tai dažniausiai yra paprasta sąlyga pavyzdžiui (*i < 20*).
- *reiškinys2* tai kažkoks reiškinys, kuris keičia ciklo kintamojo reikšmę. Dažniausiai koks nors paprastas reiškinys pavyzdžiui (*i++, i *= 20, i /=0.5*).

Šis ciklo for() pavyzdys suskaičiuoja dvidešimt sinuso ir kosinuso reikšmių intervale nuo 0 iki 1.57 radijanų ir jas atspausdina.

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

int main(int argc, char* argv[]) {
double  angle, x, y,
        angl=1.57,              // kampas iki kurio skaiciuti
        cnt=20;                 // iteraciju skaicius
int     i=0;

    printf(" Nr |  angle  |  sin(x) |  cos(x)\n");
    printf(" ---+---------+---------+---------\n");
```

```
    for ( angle = 0; angle <= angl; angle += angl/cnt ) {
        x = sin(angle);          // apskaiciuojamas sinusas
        y = cos(angle);          // apskaiciuojamas cosinusas
        printf("%3d |  %1.4f |  %1.4f |  %1.4f\n", i++,angle, x, y);
    }
    printf("\n");
    system("Pause");
    return 0;
}
```

```
 Nr |  angle  |  sin(x) |  cos(x)
----+---------+---------+--------
  0 |  0.0000 |  0.0000 |  1.0000
  1 |  0.0785 |  0.0784 |  0.9969
  2 |  0.1570 |  0.1564 |  0.9877
  3 |  0.2355 |  0.2333 |  0.9724
  4 |  0.3140 |  0.3089 |  0.9511
  5 |  0.3925 |  0.3825 |  0.9240
  6 |  0.4710 |  0.4538 |  0.8911
  7 |  0.5495 |  0.5223 |  0.8528
  8 |  0.6280 |  0.5875 |  0.8092
  9 |  0.7065 |  0.6492 |  0.7606
 10 |  0.7850 |  0.7068 |  0.7074
 11 |  0.8635 |  0.7601 |  0.6498
 12 |  0.9420 |  0.8087 |  0.5882
 13 |  1.0205 |  0.8524 |  0.5229
 14 |  1.0990 |  0.8908 |  0.4545
 15 |  1.1775 |  0.9237 |  0.3832
 16 |  1.2560 |  0.9509 |  0.3096
 17 |  1.3345 |  0.9722 |  0.2341
 18 |  1.4130 |  0.9876 |  0.1571
 19 |  1.4915 |  0.9969 |  0.0792
 20 |  1.5700 |  1.0000 |  0.0008

Press any key to continue . . .
```

12.4 break;

Šį operatorių jau matėme, kai kalbėjome apie *switch* operatorių, jo paskirtis yra lygiai tokia pat. Operatorius *break* nutraukia ciklo vykdymą, bet kurioje ciklo iteracijoje ir bet kurioje veiksmo vietoje. Pavyzdžiui:

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

int main(int argc, char* argv[])
{
double  m=1E11, n;
int     a=20, i, d;

    printf (" Nr |  /  |  =\n");
    printf (" --------------------------\n");
    for (i = 1; i <= 20; i++) {
        d = (a-i*2);
        /* kai kintamasis d pasidaro lygus nuliui
           ciklas nutraukiamas */
        if (d == 0) break;
        m /= d;
        printf ("%3d | %3d | %15.3f\n", i, d, m);
    }
    printf("\n\n");
    system("Pause");
    return 0;
}
```

ciklas bus nutrauktas kai *(a–i*2)* bus lygus *0*, nors ciklas yra iki *i <= 20*.

```
Nr |  /  |  =
 ---------------------------
  1 |  18 |  5555555555.556
  2 |  16 |   347222222.222
  3 |  14 |    24801587.302
  4 |  12 |     2066798.942
  5 |  10 |      206679.894
  6 |   8 |       25834.987
  7 |   6 |        4305.831
  8 |   4 |        1076.458
  9 |   2 |         538.229

Press any key to continue . . .
```

12.5 continue;

Programuojant reikia ne tik baigti ciklą ankščiau laiko, bet ir tęsti nevykdant visų vėliau aprašytų veiksmų. Tam yra naudojamas operatorius *continue*. Pavyzdžiui ciklas programoje:

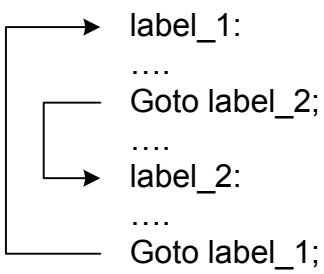
```
#include <stdlib.h>
#include <stdio.h>

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

    int    i=0;

    for (i = -5; i <= 5; i++) {
        if (i == 0)                    // kai kintamasis i lygus nuliui
            continue;
        printf ("%2d ", i);
    }
    printf("\n\n");
    system("Pause");
    return 0;
}
```

```
-5 -4 -3 -2 -1  1  2  3  4  5

Press any key to continue . . .
```

Matome, kad (0) neatspausdintas, nes kai (*i == 0*) yra įvykdoma *continue* komanda, kuri perduoda programos vykdymą į ciklo pradžią ir funkcija printf() nėra vykdoma.

12.6 goto label:

Besąlyginio perėjimo operatorius *goto* perduoda programos vykdymą į pažymėtą programos vietą.



Šis operatorius C programuotojų yra nemėgstamas ir nenaudojamas. O jo naudojimas laikomas blogu programavimo stiliumi. Teisingai programuojant šio operatoriaus nereikia.

13 Raktiniai žodžiai (keywords)

Raktiniai žodžiai tai žodžiai, kurie yra rezervuoti C kalbai ir turi nustatytą prasmę. Jie negali būti naudojami, kaip identifikatoriai. Visi raktiniai žodžiai turi būti rašomi mažosiomis raidėmis. Čia pateikiami 63 dažniausiai naudojami C ir C++ raktiniai žodžiai.

asm	auto	bool	break	case	catch	char
class	const	const_cast	continue	default	delete	do
double	dynamic_cast	else	enum	explicit	export	extern
false	float	for	friend	goto	if	inline
int	long	mutable	namespace	new	operator	private
protected	public	register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	struct	switch	template	this
throw	true	try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile	wchar_t	while

14 Funkcijos

Funkcija yra blokas arba modulis, kuris programoje atlieka tam tikrą veiksmą. Ji yra priemonė izoliuoti vieną programos bloką ar modulį nuo kitų nepriklausomų programos dalių. Tai suteikia C kalbai dvi svarbias galimybes:

- padaryti dalį programos nepriklausomą nuo kito programos kodo ir pavesti jai atlikti tam tikrą užduotį;
- padaryti dalį programos, kuri, programos kode, gali būti panaudota kiek nori kartų be jokių pakeitimų.

Funkcijos, kaip ir kintamojo, vardas turi prasidėti lotyniškos abėcėlės raide arba pabraukimo (underscore (_)) simboliu, o kiti simboliai gali būti parinkti iš šių grupių:

a ... z	(mažosios lotyniškos raidės nuo **a** iki **z**)
A ... Z	(didžiosios lotyniškos raidės nuo **A** iki **Z**)
0 ... 9	(skaičiai nuo **0** iki **9**)
_	(pabraukimas) (underscore)

Funkcija gali grąžinti reikšmę, arba negrąžinti reikšmės. Funkcijos grąžinančios char tipo reikšmę pavyzdys:

```
char leter_only (char c)
{
    if ((c>='a' && c<='z') || (c>='A' && c<='Z'))
        return (c);
    else
        return (0);
}
```

Šifunkcija dar gali būti vadinama *char* funkcija arba *char* tipo funkcija, todėl kad ji grąžina *char* tipo reikšmę.

Štai kitas, funkcijos negrąžinančios reikšmės, pavyzdys:

```
void beep (int tone,int duration)
{
    ....
// funkcijos atliekama programos dalis
}
```

Funkcijai gali būti perduodami argumentai arba ji gali neturėti argumentų.

Funkcijos parametrai, kurie yra deklaruojami funkcijos deklaravimo metu yra privalomi. Pagal tai kompiliatorius tikrina ar teisingai yra naudojama funkcija. Tokios funkcijos turi fiksuotą parametrų skaičių, tačiau gali būti ir funkcijų su kintamu parametrų skaičiumi.

C kalboje funkcijos parametrus galima deklaruoti dvejopai:

```
int function1(char *name, long idx, double price, int quantity)
{
    int status;
    ...
// funkcijos atliekama programos dalis
    ...

    return (status);
}
```

Skyrelyje išvardintos funkcijos turi fiksuotą parametrų skaičių:

`char leter_only (char c)` turi 1 parametrą;

`void beep (int tone,int duration)` – 2 parametrus;

`int function1(char *name, long idx, double price, int quantity)` – 4.

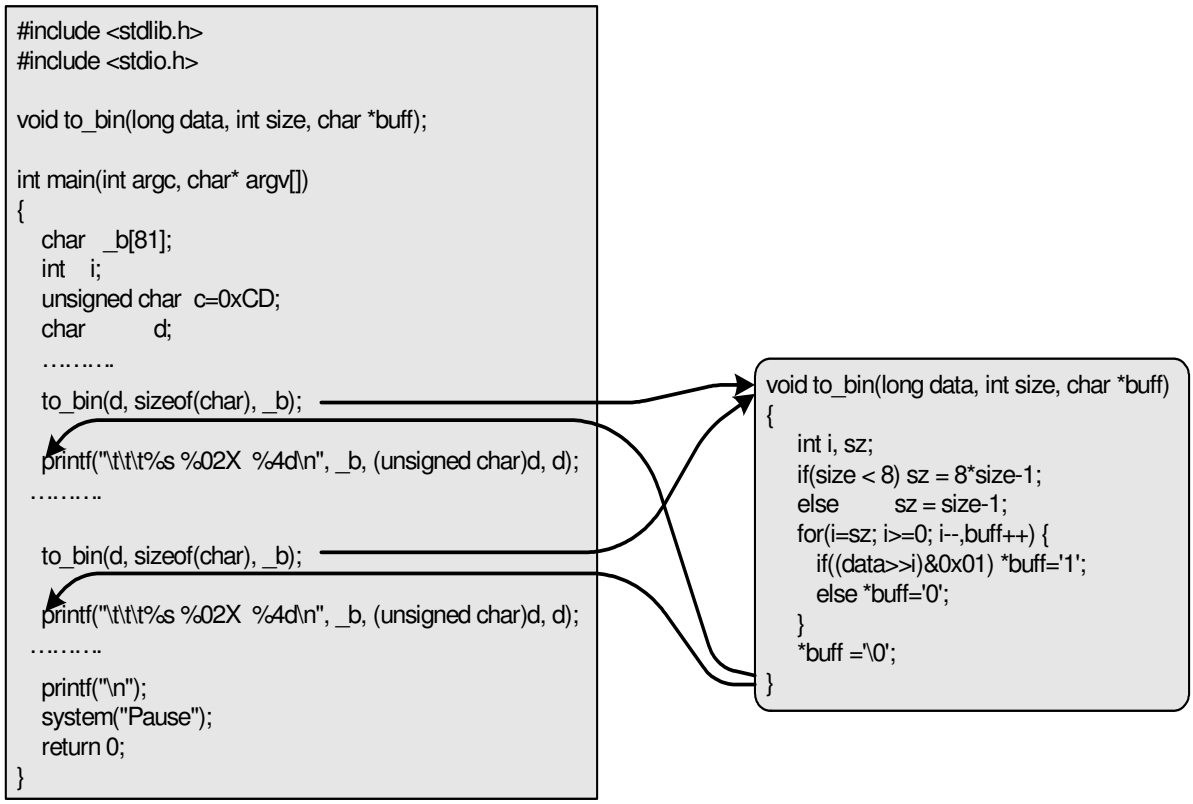
Tačiau, kaip jau buvo minėta, yra funkcijų turinčių kintamą parametrų skaičių arba parametrai gali būti skirtingų tipų. Pavyzdžiui jau daug kartų naudota funkcija printf(), kuri yra štai taip deklaruota antraščių faile <stdio.h>

```
int printf(const char * __format, ...);
```

prisimenate, kad printf() funkcijai galima perduoti bet kokį parametrų skaičių. Būtent todėl ir negalima nurodyti nei parametrų kintamųjų tipų nei skaičiaus.

14.1 Funkcijos iškvietimas

Kaip jau buvo minėjau, funkcija yra nepriklausoma programos dalis, kuri programoje gali būti panaudota tiek kartų kiek reikia be jos kodo pakeitimo. Panagrinėsime, kaip programa iškviečia funkciją (14.1 pav.). Funkcija *to_bin()* yra iškviečiama su tam tikrais argumentais, kurie kiekvieną kartą gali turėti skirtingas reikšmes. Matome, kad programos valdymas yra perduodamas funkcijai, o funkcijai pasibaigus yra grąžinamas į sekančią programos eilutę po funkcijos iškvietimo.



14.1 pav. Funkcijos iškvietimas

14.2 return (…)

Operatorius *return* priverčia funkciją užsibaigti ir grąžinti reikšmę, funkciją iškvietusiai funkcijai ar programai. Šio operatoriaus neturi būti funkcijose, kurios negrąžina reikšmės (*void func()*). Operatoriaus *return* sintaksė yra tokia:

```
      return konstanta;
arba
      return (reiškinys);
```

Pavyzdžiui:

```
int is_more (int x, int y) {
    if (x > y)
        return (1);
    else
        return (0);
}
```

arba

```
int is_max (int x, int y) {
    if (x > y)
        return (x);
    else
        return (y );
}
```

arba

```
int abs_diff (int x, int y) {
    if (x > y)
        return (x-y);
    else
        return (y-x);
}
```

15 Programavimo priemonės

15.1 C kalbos kompiliatoriai

C kalbos kompiliatorių yra labai daug ir labai įvairių. Bet visi jie turi tam tikras vienodas sudedamąsias dalis. Pakalbėsime apie pagrindines. Funkciniu požiūriu atskiros dalys yra laikomos atskirose kataloguose (directory). Aptarsime programas ir kitus failus (file) esančius kataloguose **bin**, **lib** ir **include**.

Kataloge **bin** yra C kompiliatoriaus ir papildomos programos. Programų pavadinimai gali būti gan įvairūs:

cpp, cpp32, bcc, bcpp32, gcc, cc ir t.t. – taip yra vadinami C kompiliatoriai;
link, ilink, mink, glink ir t.t. – taip vadinamos komponavimo programos;
asm, tasm, masm, iasm, gasm ir t.t. – taip vadinami transliatoriai iš asemblerio;
lib, mlib, ar, mplib ir t.t. – taip vadinamos bibliotekų surinkimo programos;
make – taip vadiname kompiliavimo automatizavimo priemonė.

Šiame kataloge paprastai būna daugiau programų, bet tai kiekvieno programavimo paketo ypatybės.

Programuojant kur kas svarbesnės už programas yra antraštės ir bibliotekos.

15.2 Antraštės (headers) ir bibliotekos (libraries)

Jau pastebėjote, kad rašant programas, programos teksto pradžioje visada yra tokios ar panašios eilutės prasidedančios preprocesoriaus direktyva *#include*:

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

tai antraščių įtraukimas į programos tekstą.

Jau ne kartą buvo minėta, kad C kalboje galima naudotis tik objektais (kintamaisiais, konstantomis, funkcijomis ir t.t.), kurie yra deklaruoti. Tikriausiai atkreipėte dėmesį, kad pačių parašytos funkcijos, yra deklaruojamos arba rašomos pirmiau negu *main()* funkcija. Nors dar nekalbėjome apie C kalbos bibliotekas ir funkcijas esančias šiose bibliotekose, tačiau jau naudojome funkciją *printf()*, aprašytą <stdio.h> antraštėje, *getch()*, aprašytą <conio.h>, *system()* aprašytą <stdlib.h>.

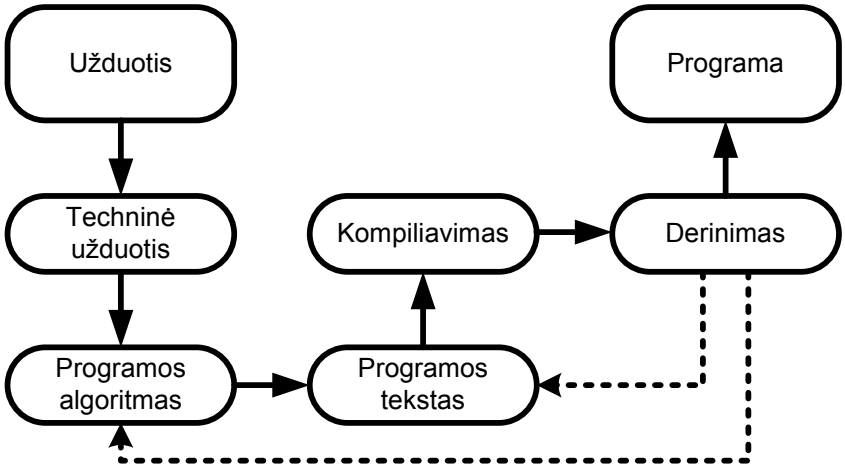
C kalbos bibliotekų antraščių failai dažniausiai yra kompiliatoriaus **include** kataloge. Antraščių failųų vardai dažniausiai atitinka bibliotekų vardus tik skiriasi plėtiniais: *.h ir *.hcc – antraštėms ir *.l ir *.lib – bibliotekoms.

Antraščių failai yra tekstiniai, juos galima pažiūrėti su bet kokiu teksto redaktoriumi. Tačiau redaguoti galima tik su simboliniais redaktoriais, nes tokie redaktoriai kaip MS Word gali sugadinti. Antraštės faile yra surašytos makro komandos, konstantiniai kintamieji, funkcijų deklaracijos, struktūrų aprašymai ir t.t.

Bibliotekų failai yra dvejetainiai ir jų redaguoti negalima, o ir žiūrėti nėra reikalo. Jie yra sudaryti iš funkcijų objektinių modulių, naudojant specialias programas bibliotekininkus. Su šiomis programomis galima sudaryti savo bibliotekas, išimti, įdėti ar pakeisti objektinius modulius.

15.3 Programavimo procesas

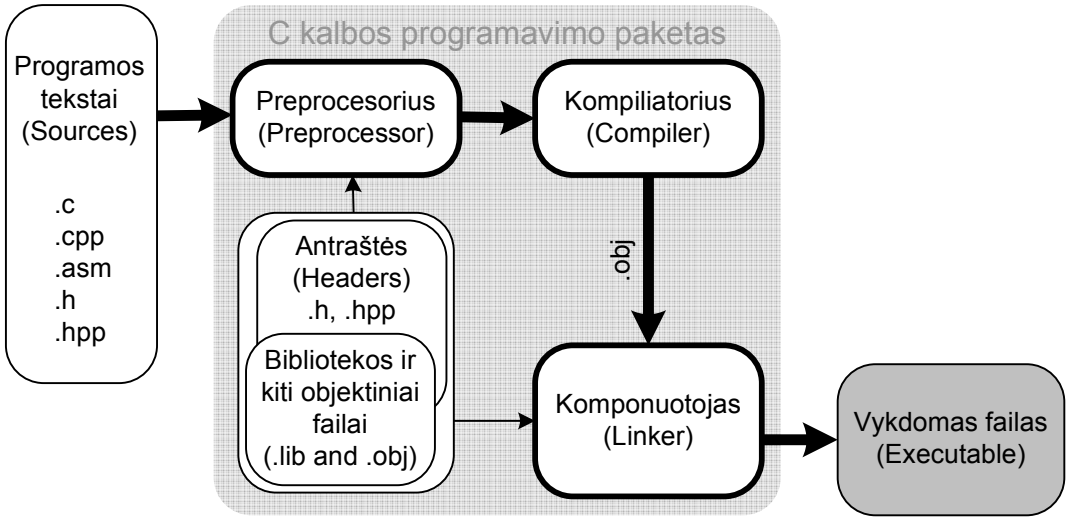
Programos sukūrimas yra gan ilgas ir sudėtingas procesas, kuris schematiškai pavaizduotas 5 paveiksle. Kaip ir bet kuris darbas jis prasideda nuo užduoties, kurioje turi būti išaiškinta ką reikia padaryti. Išsiaiškinus ką reikia padaryti, smulkiai aprašomi visi įeinantys duomenys ar signalai, valdymas ir išeinantys duomenys ar signalai.



15.1 pav. Programos kūrimo etapai

Ši informacija sudaro techninę užduotį, kuri yra pagrindas sudarant programos veikimo algoritmą. Sudėtingų programų algoritmų sudarymui naudojamos įvairios programinės priemonės, kurios naudoja UML (Universal Modeling Language) kalbą.
Sudarant algoritmą, būtina išanalizuoti ne tik darbinius duomenis ir signalus, bet ir avarines būsenas, kurioms susidarius programos veikimas neturi sugriūti. Avarinė būsena turi būti atpažinta, programa turi pranešti apie avariją ir jos priežastį ir nesugriuvusi tęsti darbą.

Sudarius algoritmą yra sprendžiama kokia programavimo kalba rašyti programą. Pasirinkus programavimo kalbą, pagal algoritmą yra parašomas programos tekstas. Programų tekstų rašymui galima naudoti ir paprastą teksto redaktorių, tačiau yra daugybė redaktorių skirtų programų



15.2 pav. Programos kompiliavimas

rašymui. Šie teksto redaktoriai turi įvairių priemonių palengvinančių programos teksto rašymą nuo paprasčiausių, kurios paryškina sintaksę, iki funkcijų sintaksės, kintamųjų ir duomenų struktūrų tikrinimo ir kitų galimybių.

Parašytas programos tekstas yra kompiliuojamas. Kompiliavimo procesas schematiškai pavaizduotas 6 paveiksle. Kai reikia sukompiliuoti programą, kuriuos programos ir antraščių tekstai yra keliuose failuose, yra naudojama **make** programa. Kompiliavimo procesas yra aprašomas faile, kurios vardas dažniausiai yra **Makefile**, jame yra surašyta programos dalių kompiliavimo ir surinkimo procedūrų eilės tvarka.

Jei rašant programą yra padaryta klaidų (dažniausiai taip ir būna), kompiliatorius apie tai praneša. Kompiliavimas kartojamas tiek kartų kiek reikia, kol programa sukompiliuojama sėkmingai. Kad programa sukompiliuojama sėkmingai, dar nereiškia, kad ji teisingai veikia.

Kai programa sukompiliuojama, pradedamas programos derinimas. Programos derinimas yra programos veikimo tikrinimas visiems galimiems įėjimo duomenims ir signalams, o taip pat visiems duomenims ir signalams, kurie gali atsirasti avariniame režime. Jei reikia yra daromi pakeitimai programos tekste ar algoritme.

Kompiliavimas iš tikrųjų nėra viena operacija. Jau minėjome, kad sudėtingoms programoms kompiliuoti yra naudojama **make** programa, kuri skaito nurodymus iš **Makefile** ir iškviečia reikalingas programas veiksmams atlikti. Pagrindiniai veiksmai ir programos, kurios vykdo šiuos veiksmus yra:

- preprocesorius (preprocessor) skaito programos tekstą, ieškodamas jam skirtų komandų, kurios prasideda simboliu (#), parašytu būtinai prie kairiojo krašto (be tarpų ir tabuliacijos). Pagrinde jis į programos tekstą įtraukia visus išvardintus antraščių failus. Tačiau tarp preprocesoriaus komandų yra ir kompiliatoriaus valdymo komandos arba taip vadinamos sąlyginės kompiliacijos komandos.
- kompiliatorius kompiliatorius (compiler) iš programos teksto padaro objektinį failą, kurio plėtinys dažniausiai yra *.obj*. Tai dar nėra programa, kuri gali veikti kompiuteryje, bet tai jau nebe tekstinis o dvejetainis failas. Kad objektinį failą paversti programa reikalingas komponuotojas.
- komponuotojas komponuotojas (linker) Iš objektinių failų surenka programos failą, kuris gali būti vykdomas kompiuteryje.

Tikriausiai jau matėte failus su plėtiniu *.lib*, juos padaro bibliotekininkas, programa, kuri apjungia kelis objektinius failas į vieną su *.obj* plėtiniu. Tai daroma tam, kad nereikėtų nurodyti dešimčių ar šimtų failų vardų.

Priedai

Priedas A. Simbolių kodavimo lentelė (ASCII)

Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
0	0	0	CTRL-@	32	40	20	Space	64	100	40	@	96	140	60	`
1	1	1	CTRL-A	33	41	21	!	65	101	41	A	97	141	61	a
2	2	2	CTRL-B	34	42	22	"	66	102	42	B	98	142	62	b
3	3	3	CTRL-C	35	43	23	#	67	103	43	C	99	143	63	c
4	4	4	CTRL-D	36	44	24	$	68	104	44	D	100	144	64	d
5	5	5	CTRL-E	37	45	25	%	69	105	45	E	101	145	65	e
6	6	6	CTRL-F	38	46	26	&	70	106	46	F	102	146	66	f
7	7	7	CTRL-G	39	47	27	'	71	107	47	G	103	147	67	g
8	10	8	CTRL-H	40	50	28	(	72	110	48	H	104	150	68	h
9	11	9	CTRL-I	41	51	29	)	73	111	49	I	105	151	69	i
10	12	A	CTRL-J	42	52	2A	*	74	112	4A	J	106	152	6A	j
11	13	B	CTRL-K	43	53	2B	+	75	113	4B	K	107	153	6B	k
12	14	C	CTRL-L	44	54	2C	,	76	114	4C	L	108	154	6C	l
13	15	D	CTRL-M	45	55	2D	-	77	115	4D	M	109	155	6D	m
14	16	E	CTRL-N	46	56	2E	.	78	116	4E	N	110	156	6E	n
15	17	F	CTRL-O	47	57	2F	/	79	117	4F	O	111	157	6F	o
16	20	10	CTRL-P	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	CTRL-Q	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	CTRL-R	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	CTRL-S	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	CTRL-T	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	CTRL-U	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	CTRL-V	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	CTRL-W	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	CTRL-X	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	CTRL-Y	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1A	CTRL-Z	58	72	3A	:	90	132	6A	Z	122	172	7A	z
27	33	1B	CTRL-[	59	73	3B	;	91	133	6B	[	123	173	7B	{
28	34	1C	CTRL-\	60	74	3C	<	92	134	6C	\	124	174	7C	|
29	35	1D	CTRL-]	61	75	3D	=	93	135	6D	]	125	175	7D	}
30	36	1E	CTRL-^	62	76	3E	>	94	136	6E	^	126	176	7E	~
31	37	1F	CTRL-_	63	77	3F	?	95	137	6F	_	127	177	7F	DEL
Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char
128	200	80	Ç	160	240	A0	á	192	300	C0	└	224	340	E0	α
129	201	81	ü	161	201	A1	í	193	301	C1	┴	225	341	E1	ß
130	202	82	é	162	202	A2	ó	194	302	C2	┬	226	342	E2	Γ
131	203	83	â	163	203	A3	ú	195	303	C3	├	227	343	E3	π
132	204	84	ä	164	204	A4	ñ	196	304	C4	─	228	344	E4	Σ
133	205	85	à	165	205	A5	Ñ	197	305	C5	┼	229	345	E5	σ
134	206	86	å	166	206	A6	ª	198	306	C6	╞	230	346	E6	µ
135	207	87	ç	167	207	A7	º	199	307	C7	╟	231	347	E7	τ
136	210	88	Ê	168	210	A8	¿	200	310	C8	╚	232	350	E8	Φ
137	211	89	Ë	169	211	A9	⌐	201	311	C9	╔	233	351	E9	Θ
138	212	8A	È	170	212	AA	¬	202	312	CA	╩	234	352	EA	Ω
139	213	8B	Ï	171	213	AB	½	203	313	CB	╦	235	353	EB	δ
140	214	8C	Î	172	214	AC	¼	204	314	CC	╠	236	354	EC	∞
141	215	8D	ì	173	215	AD	¡	205	315	CD	═	237	355	ED	φ
142	216	8E	Ä	174	216	AE	«	206	316	CE	╬	238	357	EE	ε
143	217	8F	Å	175	217	AF	»	207	317	CF	╧	239	350	EF	∩
144	220	90	É	176	220	B0	░	208	320	D0	╨	240	361	F0	≡
145	221	91	æ	177	221	B1	▒	209	321	D1	╤	241	362	F1	±
146	222	92	Æ	178	222	B2	▓	210	322	D2	╥	242	363	F2	≥
147	223	93	ô	179	223	B3	│	211	323	D3	╙	243	364	F3	≤
148	224	94	ö	180	224	B4	┤	212	324	D4	╘	244	365	F4	⌠
149	225	95	ò	181	225	B5	╡	213	325	D5	╒	245	366	F5	⌡
150	226	96	û	182	226	B6	╢	214	326	D6	╓	246	367	F6	÷
151	227	97	ù	183	227	B7	╖	215	327	D7	╫	247	360	F7	≈
152	230	98	ÿ	184	230	B8	╕	216	330	D8	╪	248	370	F8	°
153	231	99	Ö	185	231	B9	╣	217	331	D9	┘	249	371	F9	∙
154	232	9A	Ü	186	232	BA	║	218	332	DA	┌	250	372	FA	·
155	233	9B	¢	187	233	BB	╗	219	333	DB	▉	251	373	FB	√
156	234	9C	£	188	234	BC	╝	220	334	DC	▄	252	374	FC	ⁿ
157	235	9D	¥	189	235	BD	╜	221	335	DD	▌	253	375	FD	²
158	236	9E	Pts	190	236	BE	╛	222	336	DE	▐	254	376	FE	■
159	237	9F	ƒ	191	237	BF	┐	223	337	DF	▀	255	377	FF	

Priedas B. Standartinės įvedimo/išvedimo funkcijos scanf() printf()

C kalboje yra aprašytos įvedimo ir išvedimo funkcijos. Be abejo galima parašyti ir savo įvedimo ir išvedimo funkcijas, tačiau dažniausiai yra patogiau pasinaudoti jau esamomis funkcijomis.

Funkcijos *scanf()*, *fscanf()*, *sscanf()*, *printf()*, *fprintf()*, *sprintf()* ir dar keletas funkcijų yra aprašytos antraščių faile *stdio.h*, o objektiniai moduliai yra bibliotekoje *stdio.lib*. Visų šių funkcijų naudojimo sintaksė yra praktiškai vienoda, o svarbiausia jų dalis formatas yra vienodas.

```
printf("tekstas ir formatų specifikatoriai", parametrai);
```

Formato specifikatoriai yra:

%[flag] [width] [.prec] [F|N|h|l|L] type

%[žymė] [plotis] [.tikslumas] [F|N|h|l|L] tipas

tipas (type)

%d	spausdina sveiką dešimtainį skaičių (int)
%ld	spausdina long tipo sveiką dešimtainį skaičių
%u	spausdina sveiką dešimtainį skaičių be ženklo
%c	spausdina vieną simbolį
%s	spausdina eilutę
%f	spausdina skaičių su slankiu (plaukiojančiu) kableliu
%e	kaip ir %f, bet eksponentine forma
%E	kaip %e, tik didžioji raidė **E** eksponentei
%g	naudoja %e arba %f, priklausomai nuo to kuris geriai
%G	kaip %f arba %e, tik didžioji raidė **E** eksponentei
%o	spausdina sveiką skaičių, kaip aštuonetainį (bazė 8)
%x	spausdina sveiką skaičių, kaip šešioliktainį (base 16)
%X	kaip ir %x, tik didžiosios **A, B, C, D, E** ir **F**
%%	spausdina simbolį %

žymė (flag)

-	lygiavimas kairėje, jei ne lygiuojama dešinėje
+	spausdinti su ženklu, jei ne spausdinamas tik minus ženklas (-) neigiamiems skaičiams
tarpas	prieš skaičių spausdinamas tarpas vietoje ženklo plius (+), neigiami skaičiai spausdinami visada su minus (-) ženklu.

plotis (width)

n	mažiausiai n simbolių bus atspausdinta. Jei spausdinimui yra mažiau simbolių, trūkstamų simbolių vietoje bus spausdinami tarpai.
0n	priekyje trūkstami simboliai pakeičiami simboliu nulis (0)

tikslumas (precision)

(nenurodytas)	iš anksto nustatytas (default) tikslumas 1 - d, i, o, u, x ir X tipams 6 - f, e ir E tipams visi reikšmingi skaičiai - g ir G tipams spausdina iki pirmojo NULL simbolio - s tipui
.0	tik sveikoji dalis e, E ir f tipams
.n	spausdina n reikšmių po kablelio, jei reikšmių yra daugiau jos yra suapvalinamos arba atmetamos. Tipui s - kiek daugiausiai simbolių galima spausdinti.

pločio modifikatorius

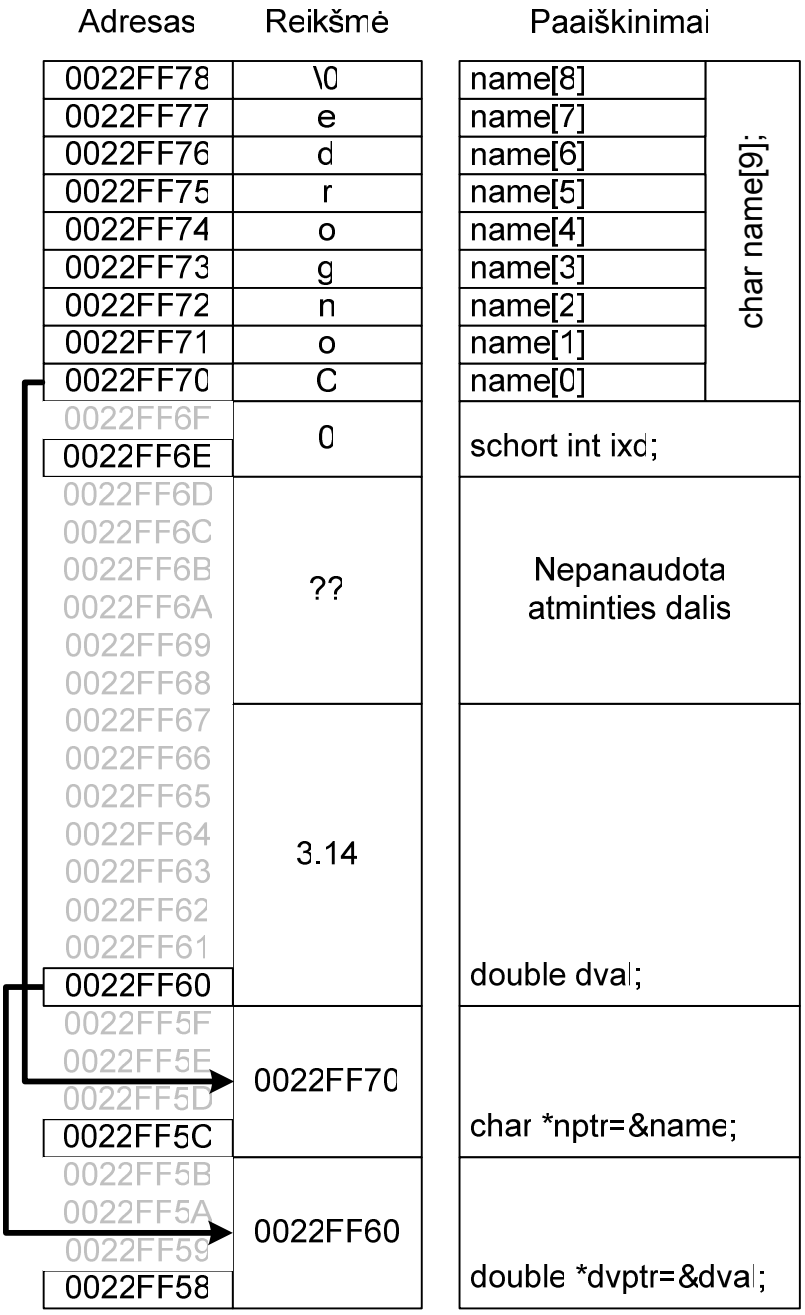
F	argumentas nuskaitomas kaip tolima rodyklė (far pointer)
N	argumentas nuskaitomas, kaip artima rodyklė (near pointer)
h	argumentas interpretuojamas kaip *short int* tipams d, o, u, x ir X
l	argumentas interpretuojamas kaip *long int* tipams d, o, u, x ir X, arba kaip *double* e, E ir f tipams.
L	argumentas interpretuojamas kaip long double tipams e, E, f, g ir G.

Programos pavyzdys, demonstruojantis formatuotą išvedimą, naudojant *printf()* funkcija.

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

#define I 555
#define R 5.5
#define S "String"

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

    int i,j,k,l;
    char buf[7];
    char *prefix = buf;
    char tp[100];

    printf("prefix |    6d |    6o |      8x |       10.2e |"
           "     10.2f |      8s| 5.3s|\n");
    printf("-------+-------+-------+---------+-------------+"
           "-----------+--------+-----+\n");

    strcpy(prefix, "%");

    for( i = 0; i < 2; i++) {
        for( j = 0; j < 2; j++) {
            for( k = 0; k < 2; k++) {
                for( l = 0; l < 2; l++) {
                    if(i==0) strcat(prefix, "-");
                    if(j==0) strcat(prefix, "+");
                    if(k==0) strcat(prefix, "#");
                    if(l==0) strcat(prefix, "0");
                    printf("%6s |",prefix);

                    strcpy(tp, prefix);
                    strcat(tp, "6d |");
                    strcat(tp, prefix);
                    strcat(tp, "6o |");
                    strcat(tp, prefix);
                    strcat(tp, "8x |");
                    printf(tp, I, I, I);

                    strcpy(tp, prefix);
                    strcat(tp, "12.2e |");
                    strcat(tp, prefix);
                    strcat(tp, "10.2f |");
                    printf(tp, R, R);

                    strcpy(tp, prefix);
                    strcat(tp, "8s|");
                    strcat(tp, prefix);
                    strcat(tp, "5.3s|");
                    printf(tp, S, S);
                    printf("  \n");
                    strcpy(prefix, "%");
                }
            }
        }
    }
    printf("\n");
    system("Pause");
}
```

```
prefix |    6d |    6o |      8x |       10.2e |     10.2f |      8s| 5.3s|
-------+-------+-------+---------+-------------+-----------+--------+-----+
 %-+#0 |+555   |01053  |0x22b    |+5.50e+000   |+5.50      |String  |Str  |
  %-+# |+555   |01053  |0x22b    |+5.50e+000   |+5.50      |String  |Str  |
  %-+0 |+555   |1053   |22b      |+5.50e+000   |+5.50      |String  |Str  |
   %-+ |+555   |1053   |22b      |+5.50e+000   |+5.50      |String  |Str  |
  %-#0 |555    |01053  |0x22b    |5.50e+000    |5.50       |String  |Str  |
   %-# |555    |01053  |0x22b    |5.50e+000    |5.50       |String  |Str  |
   %-0 |555    |1053   |22b      |5.50e+000    |5.50       |String  |Str  |
    %- |555    |1053   |22b      |5.50e+000    |5.50       |String  |Str  |
  %+#0 |+00555 |001053 |0x00022b |+005.50e+000 |+000005.50 |00String|00Str|
   %+# |  +555 | 01053 |   0x22b |  +5.50e+000 |     +5.50 |  String|  Str|
   %+0 |+00555 |001053 |0000022b |+005.50e+000 |+000005.50 |00String|00Str|
    %+ |  +555 |  1053 |     22b |  +5.50e+000 |     +5.50 |  String|  Str|
   %#0 |000555 |001053 |0x00022b |0005.50e+000 |0000005.50 |00String|00Str|
    %# |   555 | 01053 |   0x22b |   5.50e+000 |      5.50 |  String|  Str|
    %0 |000555 |001053 |0000022b |0005.50e+000 |0000005.50 |00String|00Str|
     % |   555 |  1053 |     22b |   5.50e+000 |      5.50 |  String|  Str|

Press any key to continue . . .
```

Priedas C. Kompiuterio atminties paskirstymo pavyzdys



Priedas D. Programų pavyzdžiai

D.1 Kintamųjų deklaracija ir inicializacija (reikšmių priskyrimas)

```
/*-------------------------------------------
      declaration.c
      Contains some variable declaration and
        initialization examples
      ©Siauliai University
  -------------------------------------------*/

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

int main(int argc, char* argv[])
{
    int      i, j, k;

    /* Simple */
    char    _digit = 8;
    char    _char = 'A';
    short   _short_value;
    int     _int_var1, _int_var2;
    long    l_val=0;
    float   _e = 2.7;
    double  _pi = 3.1415926;
    double  small = 1.18E-47;

    /* Arrays*/
    char    d_array[6] = {10, 11, 12, 13, 14, 15};
    char    c_array1[9] = {"Concorde"};
    char    c_array2[9] =
            {'C', 'o', 'n', 'c', 'o', 'r', 'd', 'e', '\0'};
    char    c_array3[3][16]=
            {"Concorde", "Boing 747", "Aerobus 370"};
    char    c_array4[3][16]= {
            {'C','o','n','c','o','r','d','e'},
            {'B','o','i','n','g',' ','7','4','7'},
            {'A','e','r','o','b','u','s',' ','3','7','0'}
    };

    /* Positive and negative */
    char    _cn = -127;
    char    _cp = 127;

    /* decimal, octal and hexadecimal */
    unsigned char _cd = 65;   // decimal
    unsigned char _co = 0102; // octal 4 numbers starts with 0 (zero)
    unsigned char _cx = 0x4E; // hexadecimal starts with 0x signature

    printf("Simple\n");
    printf("\t_digit: %d\n", _digit);
    printf("\t_char : %c\n", _char);
    printf("\t_e    : %f\n", _e);
    printf("\t_pi   : %10.8f\n", _pi);
    printf("\tsmall : %e\n", small);

    printf("\nArrays\n");
    printf("\t_array: ");
    for(i=0; i<6; i++)
        printf("%d ", d_array[i]); printf("\n");
    printf("\tc_array1: %s\n\tc_array2: %s\n\n", c_array1,c_array2);
    printf("\tc_array3[]: %s", c_array3[0]);
    for(i=1;i<3;i++) printf(", %s", c_array3[i]); printf("\n");
    printf("\n\tc_array4[0]: %s\n\tc_array4[1]: %s\n\tc_array4[2]: %s\n",
```

```
                        c_array4[0],c_array4[1],c_array4[2]);

    printf("\nNegative and Pozitive\n");
    printf("\t%d, %d\n", _cn, _cp);

    printf("\nSigned and Unsigned\n");
    printf("\t%4X, %2X\n", (unsigned char)_cn, _cp);
    printf("\nFormat decimal, octal and hexadecimal\n");
    printf("\tdec\toct\thex\n");
    printf("\t%3d\t%#3o\t%#3X",_cd,_cd,_cd);

    printf("\n");
    system("Pause");
    return 0;
}
```

```
Simple
        _digit: 8
        _char : A
        _e    : 2.700000
        _pi   : 3.14159260
        small : 1.180000e-047

Arrays
        _array: 10 11 12 13 14 15
        c_array1: Concorde
        c_array2: Concorde

        c_array3[]: Concorde, Boing 747, Aerobus 370

        c_array4[0]: Concorde
        c_array4[1]: Boing 747
        c_array4[2]: Aerobus 370

Negative and Pozitive
        -127, 127

Signed and Unsigned
          81, 7F

Format decimal, octal and hexadecimal
        dec     oct     hex
         65     0101    0X41

Press any key to continue . . .
```

D.2 if ... else ...

```
/*-------------------------------------------
      if_else.c
      Contains some if ... ele ... usage examples
      ©Siauliai University
  -------------------------------------------*/

#pragma argsused
#include <stdio.h>
#include <conio.h>
int main(int argc, char* argv[])
{
        int c;
        printf("\tIseiti is ciklo \'Esc\'\n\n");
        while ((c=getch()) != 0x1b) {
                if(c==0x3E)
                        printf("Daugiau            %c\n", c);
                else if(c==0x3D)
                        printf("Lygu               %c\n", c);
                else if(c==0x3C)
                        printf("Maziau             %c\n", c);
                else if(c >= 48 && c < 58)
                        printf("Skaicius           %c\n", c);
                else if(c > 64 && c <= 90)
                        printf("Didzioji lotyniska %c\n", c);
                else if(c >= 97 && c <= 122)
                        printf("Mazoji   lotyniska %c\n", c);
                else
                        printf("Neatpazintas simbolis %c %3d %02x\n", c,
                                (unsigned char)c, (unsigned char)c);

        }
        return 0;
}
```

```
        Iseiti is ciklo 'Esc'

Didzioji lotyniska S
Mazoji   lotyniska i
Mazoji   lotyniska a
Mazoji   lotyniska u
Mazoji   lotyniska l
Mazoji   lotyniska i
Mazoji   lotyniska a
Mazoji   lotyniska i
Neatpazintas simbolis ' '  32 20
Maziau             <
Skaicius           2
Skaicius           0
Skaicius           0
Skaicius           6
Daugiau            >
```

D.3 for(...;...;...)

```
/*-------------------------------------------
    for.c
      Contains cycle for(...;...;...) usage examples
      ©Siauliai University
  -------------------------------------------*/

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

int main(int argc, char* argv[])
{
    unsigned char     c;
    int       i, j;
    i=j=0;

    printf("  0 1 2 3 4 5 6 7 8 9 A B C D E F\n");
    for(j=0;j<=0xF;j++) {
        printf("%X ", j);
        for(i=0;i<=0xF;i++) {
            c = (unsigned char)((j*0x10)+i);
            if((c < 0x20)||(c==0x7F)||(c==0xFF))
                  c = 0xfa;
            printf("%c ", c);
        }
        printf("\n");
    }

    printf("\n");
    system("Pause");
    return 0;
}
```

```
   0 1 2 3 4 5 6 7 8 9 A B C D E F
0  · · · · · · · · · · · · · · · ·
1  · · · · · · · · · · · · · · · ·
2    ! " # $ % & ' ( ) * + , - . /
3  0 1 2 3 4 5 6 7 8 9 : ; < = > ?
4  @ A B C D E F G H I J K L M N O
5  P Q R S T U V W X Y Z [ \ ] ^ _
6  ` a b c d e f g h i j k l m n o
7  p q r s t u v w x y z { | } ~ ·
8  Ç ü é â ä à å ç ê ë è ï î ì Ä Å
9  É æ Æ ô ö ò û ù ÿ Ö Ü ¢ £ ¥ ₧ ƒ
A  á í ó ú ñ Ñ ª º ¿ ⌐ ¬ ½ ¼ ¡ « »
B  ░ ▒ ▓ │ ┤ ╡ ╢ ╖ ╕ ╣ ║ ╗ ╝ ╜ ╛ ┐
C  └ ┴ ┬ ├ ─ ┼ ╞ ╟ ╚ ╔ ╩ ╦ ╠ ═ ╬ ╧
D  ╨ ╤ ╥ ╙ ╘ ╒ ╓ ╫ ╪ ┘ ┌ █ ▄ ▌ ▐ ▀
E  α ß Γ π Σ σ µ τ Φ Θ Ω δ ∞ φ ε ∩
F  ≡ ± ≥ ≤ ⌠ ⌡ ÷ ≈ ° ∙ · √ ⁿ ² ■ ·

Press any key to continue . . .
```

D.4 C kalbos kintamieji ir masyvai

```
/*-------------------------------------------
      vars.c
      Contains some variable usage examples
      ©Siauliai University
  -------------------------------------------*/

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

int main(int argc, char *argv[])
{
int i_array[5] = {0, 1, 2, 3, 4};              // penkios skaitines reiksmes
long l_arr[3] = {123, 234, 345};               // trys skaitines reiksmes
char weak_day[7][3]=
      {"Su","Mo","Tu","We","Th","Fr","Sa"};    // septynios simbolinės eil.
int i,j;

    for(i=0;i<5;i++) {
        // spausdinamas i_array masyvo elementas
        printf("%d",i_array[i]);
    }
    printf("\n\n");

    for(i=0;i<7;i++) {
        // spausdinamas savaites dienos trumpinys
        printf("%s ",weak_day[i]);
    }
    printf("\n");
    for(i=0;i<7;i++) {
        // spausdinama savaites dienos trumpinio pirmoji raide
        printf("%c  ",weak_day[i][0]);
    }
    printf("\n\n");

    for(j=0;j<3;j++) {
        // spausdinamas l_arr masyvo elementas
        printf("%d  ",l_arr[i]);
    }
    printf("\n\n");

    system("PAUSE");
    return 0;
}
```

```
01234

Su Mo Tu We Th Fr Sa
S  M  T  W  T  F  S

3  3  3

Press any key to continue . . .
```

D.5 Rekursinė funkcija

```
/*-----------------------------------------
      factorial.c
      Example of recursion
      ©Siauliai University
  ----------------------------------------- */

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

double factorial(double num);

main()
{
  int num;
  printf("Faktorialo skaičiavimas\n" );
  printf("Iveskite skaiciu: " );
  scanf("%d", &num);

  printf("\n %d! = %.0f\n",num, factorial(num) );
  system("Pause");
}

double factorial(double num)
  {
  double ans=1;
  if (num == 1 ) return 1;
  ans = num * factorial(num-1);
  return ans;
}
```

```
Faktorialo skaiciavimas
Iveskite skaiciu: 24

 24! = 620448401733239410000000
```

D.6 printf() funkcija

```
/*-------------------------------------------
      Printf_usage.c
      Example of printf() function usage
      ©Siauliai University
  ------------------------------------------- */

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

int main(int argc, char *argv[])
{
int a;               /* simple integer type */
long int b;          /* long integer type */
short int c;         /* short integer type */
unsigned int d;      /* unsigned integer type */
char e;              /* character type */
float f;             /* floating point type */
double g;            /* double precision floating point */
char str[20];

    a = 1023;
    b = 2222;
    c = 123;
    d = 1234;
    e = 'X';
    f = 3.14159;
    g = 3.1415926535898;
    strcpy(str, " Simple String ");

    printf("decimal          | a = %d\n",a);        /* decimal output */
    printf("octal            | a = %o\n",a);        /* octal output */
    printf("hexadecimal      | a = %x\n",a);        /* hexadecimal output */
    printf("decimal long     | b = %ld\n",b);       /* decimal long output */
    printf("decimal short    | c = %d\n",c);        /* decimal short output */
    printf("unsigned         | d = %u\n",d);        /* unsigned output */
    printf("character        | e = %c\n",e);        /* character output */
    printf("floating         | f = %f\n",f);        /* floating output */
    printf("double float     | g = %f\n",g);        /* double float output */
    printf("\n");
    printf("simple int       | a = [%d]\n",a);      /* simple int output */
    printf("width of 7       | a = [%7d]\n",a);     /* use a field width of 7 */
    printf("justify width 7  | a = [%-7d]\n",a);    /* left justify width = 7 */
    printf("\n");
    printf("simple float     | f = [%f]\n",f);      /* simple float output */
    printf("width of 12      | f = [%12f]\n",f);    /* use field width of 12 */
    printf("3 decimal places | f = [%12.3f]\n",f);  /* use 3 decimal places */
    printf("5 decimal places | f = [%12.5f]\n",f);  /* use 5 decimal places */
    printf("left justify     | f = [%-12.5f]\n",f); /* left justify in field */
    printf("\n");
    printf("simple string    | [%s]\n", str);       /* simple string */
          /* use a field width of 20 */
    printf("width of 20      | [%20s]\n", str);
          /* left justify field width of 20 */
    printf("left justify     | [%-20s]\n", str);

    printf("\n");
    system("Pause");
}
```

```
decimal           | a = 1023
octal             | a = 1777
hexadecimal       | a = 3ff
decimal long      | b = 2222
decimal short     | c = 123
unsigned          | d = 1234
character         | e = X
floating          | f = 3.141590
double float      | g = 3.141593

simple int        | a = [1023]
width of 7        | a = [   1023]
justify width 7   | a = [1023   ]

simple float      | f = [3.141590]
width of 12       | f = [    3.141590]
3 decimal places  | f = [       3.142]
5 decimal places  | f = [     3.14159]
left justify      | f = [3.14159     ]

simple string     | [ Simple String ]
width of 20       | [      Simple String ]
left justify      | [ Simple String      ]

Press any key to continue . . .
```