

Osnove programskega inženirstva, C

Marjan Jenko

*Fakulteta za
strojništvo
Univerze v
Ljubljani*



Recenzenta: prof. ddr. Janez Žerovnik in doc. dr. Nikolaj Mole

Lektorirala: Andreja Cigale, prof. slov.

Grafična obdelava besedila: Avtor

© Univerza v Ljubljani, Fakulteta za strojništvo, 2014

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004.43(075.8)(0.034.2)

JENKO, Marjan, 1961-

Osnove programskega inženirstva, C [Elektronski vir] /

Marjan Jenko. - El. knjiga. - Ljubljana : Fakulteta za strojništvo,
2014

Način dostopa (URL):

<http://www2.arnes.si/~mjenko9/OPI-C/OPI-C.pdf>

ISBN 978-961-6536-80-6 (pdf)

276199168



9 789616 536806

Kazalo

1	Uvod	1
1.1	Programsko inženirstvo	1
1.2	Vgradni sistem.....	6
1.2.1	Strojna oprema vgradnega sistema	9
1.2.2	Programska oprema vgradnega sistema.....	13
1.3	Učenje jezika C.....	14
2	Prvi programi – prevajanje in izpis.....	15
2.1	Naloge.....	16
2.1.1	N: Dober dan, svet	16
2.1.2	N: Navodilo programerju.....	16
2.1.3	N: <i>printf()</i> , vrnjena vrednost	17
2.1.4	N: Osebna vizitka	17
2.2	Rešitve (R).....	18
2.2.1	R: Dober dan, svet	18
2.2.2	R: Navodilo programerju.....	19
2.2.3	R: <i>printf()</i> , vrnjena vrednost	20
2.2.4	R: Osebna vizitka.....	21
3	Spremenljivke, podatkovni tipi in aritmetični izrazi	22
3.1	Naloge.....	23
3.1.1	N: <i>scanf()</i> , <i>printf()</i>	23
3.1.2	N: Kalkulator	24
3.1.3	N: Polinom.....	24
3.1.4	N: Ulomek z eksponenti	25
3.1.5	N: Obseg in površina kroga	25

3.1.6	N: Vsota kotov.....	26
3.1.7	N: Preračun iz °F v °C	26
3.1.8	N: Pretvorba iz radianov v stopinje	27
3.1.9	N: Obrat števk.....	27
3.2	Rešitve (R).....	28
3.2.1	R: scanf(), printf().....	28
3.2.2	R: Kalkulator	29
3.2.3	R: Polinom.....	30
3.2.4	R: Ulomek z eksponenti.....	31
3.2.5	R: Obseg in površina kroga	33
3.2.6	R: Vsota kotov	34
3.2.7	R: Preračun iz °F v °C	36
3.2.8	R: Pretvorba iz radianov v stopinje.....	37
3.2.9	R: Obrat števk.....	38
4	Programske zanke.....	39
4.1	Naloge.....	40
4.1.1	N: Izpis ASCII znakov, [0 .. 255].....	40
4.1.2	N: Izpis ASCII znakov iz intervala.....	40
4.1.3	N: Izpis abecede velikih črk	41
4.1.4	N: Izpis abecede malih črk	41
4.1.5	N: <i>getchar()</i> , <i>putchar()</i>	41
4.1.6	N: 12 faktoriel.....	42
4.1.7	N: Tabela sinus[0..2π].....	43
4.1.8	N: Integral <i>sin(x)</i>	43
4.1.9	N: Integral polinoma.....	44
4.1.10	N: Fibonaccijevo zaporedje	44
4.1.11	N: Piramida.....	45

4.1.12	N: Poštevanka $10 * 10$	45
4.1.13	N: Vsota števk	46
4.1.14	N: e^x kot vrsta do (člen $< \varepsilon$)	46
4.2	Rešitve (R).....	47
4.2.1	R: Izpis ASCII znakov, [0 .. 255]	47
4.2.2	R: Izpis ASCII znakov iz intervala.....	48
4.2.3	R: Izpis abecede velikih črk.....	49
4.2.4	R: Izpis abecede malih črk.....	50
4.2.5	R: <i>getchar()</i> , <i>putchar()</i>	51
4.2.6	R: 12 faktoriel	52
4.2.7	R: Tabela $\sin[0 .. 2\pi]$	53
4.2.8	R: Integral $\sin(x)$	54
4.2.9	R: Integral polinoma	55
4.2.10	R: Fibonaccijevo zaporedje	56
4.2.11	R: Piramida	57
4.2.12	R: Poštevanka $10 * 10$	58
4.2.13	R: Vsota števk.....	59
4.2.14	R: e^x kot vrsta do (člen $< \varepsilon$).....	60
5	Odločanje.....	61
5.1	Naloge.....	61
5.1.1	N: Je zadnja številka enaka »0«?	61
5.1.2	N: Celoštevilska deljivost?	62
5.1.3	N: Deljenje z realnim rezultatom.....	62
5.1.4	N: Značilke vnosov števil	63
5.1.5	N: Je leto prestopno?	64
5.1.6	N: Trikotnik a,b,c obstoja?	64
5.1.7	N: Kalkulator + - * /	65

5.1.8	N: Sklanjanje za vnos	66
5.1.9	N: Je n praštevilo?	67
5.1.10	N: Produkt ne-nič števk	67
5.1.11	N: Izpis števk z besedami	68
5.2	Rešitve (R).....	69
5.2.1	R: Je zadnja številka enaka »0«?	69
5.2.2	R: Celoštevilska deljivost?	70
5.2.3	R: Deljenje z realnim rezultatom.....	71
5.2.4	R: Značilke vnosov števil	72
5.2.5	R: Je leto prestopno?.....	74
5.2.6	R: Trikotnik a, b, c obstoja?.....	75
5.2.7	R: Kalkulator $+ - * /$	77
5.2.8	R: Sklanjanje za vnos.....	79
5.2.9	R: Je n praštevilo?.....	80
5.2.10	R: Produkt ne-nič števk	81
5.2.11	R: Izpis števk z besedami	82
6	Polja.....	84
6.1	Naloge.....	85
6.1.1	N: Vnosi in povprečje.....	85
6.1.2	N: Vnosi in dva izpisa.....	86
6.1.3	N: Kopiranje polja v polje	87
6.1.4	N: Iskanje v polju.....	88
6.1.5	N: Sprememba številske baze.....	89
6.1.6	N: Fibonaccijeva števila	89
6.1.7	N: Koliko bankovcev?	90
6.1.8	N: Transponiranje matrike.....	91
6.2	Rešitve (R).....	92

6.2.1	R: Vnosi in povprečje	92
6.2.2	R: Vnosi in dva izpisa.....	93
6.2.3	R: Kopiranje polja v polje.....	94
6.2.4	R: Iskanje v polju.....	96
6.2.5	R: Sprememba številske baze	98
6.2.6	R: Fibonaccijeva števila.....	100
6.2.7	R: Koliko bankovcev ?	101
6.2.8	R: Transponiranje matrike	102
7	Funkcije	104
7.1	Naloge.....	104
7.1.1	N: Vsota števil	104
7.1.2	N: Parameter in spremenljivka	105
7.1.3	N: Globalni spremenljivki	105
7.1.4	N: Rešitev kvadratne enačbe	106
7.1.5	N: Funkcija Največji Skupni Delitelj (NSD).....	107
7.1.6	N: Funkcija <i>Fib()</i> – Fibonaccijevo število.....	108
7.1.7	N: Funkcija s poljem.....	109
7.1.8	N: Fibonacci rekurzivno	110
7.1.9	N: Faktoriela rekurzivno.....	111
7.2	Rešitve (R).....	112
7.2.1	R: Vsota števil	112
7.2.2	R: Parameter in spremenljivka.....	113
7.2.3	R: Globalni spremenljivki.....	114
7.2.4	R: Rešitev kvadratne enačbe.....	115
7.2.5	R: Funkcija Največji Skupni Delitelj (NSD).....	119
7.2.6	R: Funkcija <i>Fib()</i> – Fibonaccijevo število	121
7.2.7	R: Funkcija s poljem.....	122

7.2.8	R: Fibonacci rekurzivno	124
7.2.9	R: Faktoriela rekurzivno	125
8	Strukture	126
8.1	Naloge.....	126
8.1.1	N: Razdalja	126
8.1.2	N: Pretečeni čas	127
8.2	Rešitve	128
8.2.1	R: Razdalja	128
8.2.2	R: Pretečeni čas	130
9	Nizi	132
9.1	Naloge.....	132
9.1.1	N: Vpis, izpis niza.....	132
9.1.2	N: Obrni niz	133
9.1.3	N: Niz v strukturi.....	133
9.1.4	Naredimo <i>najdiNiz()</i>	134
9.1.5	N: Naredimo <i>odstraniNiz()</i>	135
9.1.6	N: Naredimo <i>vstaviNiz()</i>	136
9.1.7	N: Naredimo <i>zamenjajNiz()</i>	137
9.1.8	N: Preštej besede.....	138
9.1.9	N: Sprememba velikosti črk	138
9.1.10	N: Palindrom?	139
9.1.11	N: Značilke niza.....	139
9.1.12	N: Združitev nizov	140
9.2	Rešitve	141
9.2.1	R: Vpis, izpis niza.....	141
9.2.2	R: Obrni niz	142
9.2.3	R: Niz v strukturi	143

9.2.4	R: Naredimo <i>najdiNiz()</i>	144
9.2.5	R: Naredimo <i>odstraniNiz()</i>	145
9.2.6	R: Naredimo <i>vstaviNiz()</i>	147
9.2.7	R: Naredimo <i>zamenjajNiz()</i>	149
9.2.8	R: Preštej besede.....	152
9.2.9	R: Sprememba velikosti črk.....	153
9.2.10	R: Palindrom?	154
9.2.11	R: Značilke niza.....	156
9.2.12	R: Združitev nizov	158
10	Kazalci.....	159
10.1	Naloge.....	160
10.1.1	N: Kazalci, *	160
10.1.2	N: Kazalci, * in &.....	161
10.1.3	N: Kazalci, dinamična dodelitev spremenljivke	162
10.1.4	N: Kazalec in operatorji *, &, ++	163
10.1.5	N: Dinamično narejen povezan seznam.....	164
10.1.6	N: Kazalci – parametri funkcije; statična spremenljivka.....	165
10.1.7	N: Kazalci – parametri funkcij.....	166
10.1.8	N: Kazalec na funkcijo	167
10.2	Rešitve	168
10.2.1	R: Kazalci, *	168
10.2.2	R: Kazalci, * in &.....	168
10.2.3	R: Kazalci, dinamična dodelitev spremenljivke	168
10.2.4	R: Kazalec in operatorji *, &, ++	169
10.2.5	R: Dinamično narejen povezan seznam.....	170
10.2.6	R: Kazalci - parametri funkcij; statična spremenljivka.....	172
10.2.7	R: Kazalci – parametri funkcij.....	173

10.2.8	R: Kazalec na funkcijo.....	175
11	Delo z biti	177
11.1	Naloge.....	177
11.1.1	N: Bitni operatorji.....	177
11.1.2	N: Prireditveni bitni operatorji.....	178
11.1.3	N: Velikost tipov integer, short integer, long integer	179
11.1.4	N: Iskanje bitnega vzorca	180
11.2	Rešitve	181
11.2.1	R: Bitni operatorji.....	181
11.2.2	R: Prireditveni bitni operatorji.....	183
11.2.3	R: Velikosti tipov int, short int, long int.....	185
11.2.4	R: Iskanje bitnega vzorca.....	187
12	Datotečne funkcije.....	190
12.1	Naloge.....	190
12.1.1	N: Pisanje v datoteko	190
12.1.2	N: Vpis in izpis	191
12.1.3	N: Kopiranje datoteke po bytih.....	192
12.1.4	N: Kopiranje datoteke po blokih.....	193
12.1.5	N: Sprememba v velike črke.....	194
12.1.6	N: Velikost datoteke	195
12.1.7	N: Izpis meritev	196
12.1.8	N: Izpis po n vrstic.....	198
12.2	Rešitve	200
12.2.1	R: Pisanje v datoteko	200
12.2.2	R: Vpis in izpis	201
12.2.3	R: Kopiranje datoteke po bytih.....	204
12.2.4	R: Kopiranje datoteke po blokih.....	206

12.2.5	R: Sprememba v velike črke.....	208
12.2.6	R: Velikost datoteke	209
12.2.7	R: Izpis meritev	210
12.2.8	R: Izpis po n vrstic.....	212
13	Literatura	214

e_branje:

Bralniki pdf: Adobe, Foxit, Nitro, PDF-Xchange, Sumatra.

Velikosti strani in črk sta usklajeni z navodilom Založbe FS.

1 Uvod

1.1 Programsko inženirstvo

Programsko inženirstvo je strukturiranje in izvajanje programerskih procesov od potrebe po programski rešitvi preko razvoja, uporabe, vzdrževanja programskega produkta do opustitve programske rešitve [1, str. 82-250]. Velika večina načinov strukturiranja programerskih procesov po načelih programskega inženirstva je univerzalna oziroma neodvisna od konkretnih funkcionalnih zahtev programov, njihovega namena, od platform na katerih naj delujejo ter od njihovih lastnosti in uporabnikov.

V programskem inženirstvu je uveljavljen inženirski pristop k pisanju programov. Poudarek je na zagotavljanju kakovosti programskih rešitev. To je omogočeno s sprotno verifikacijo funkcionalnosti, s strukturiranjem in z dokumentiranjem programske opreme za kasnejše vzdrževanje tekom življenjske dobe produkta ter z doseganjem rokov izdelave programske opreme ob nepreseženih stroških.

Programsko inženirstvo je interdisciplinarno področje. Obsega strukturiranje programske opreme, kodiranje, verifikacijo, timsko vodenje, razumevanje procesov, ki naj jih programska rešitev krmili in odnose z naročniki. Najbolj osnovna zahteva je, da programski produkti ne vsebujejo napak. Sprotne verifikacije funkcionalnosti zelo izboljšajo kakovost programskih rešitev. Pri razvoju računalniško krmiljenih in računalniško spremljanih distribuiranih proizvodnih procesov ter visokotehnoloških mehatronskih naprav razvoj programske opreme še vedno predstavlja visok dejavnik tveganja. Razloga sta vsaj dva: a) krmiljenje procesov in funkcionalnost naprav sta vedno bolj odvisna od programske opreme in b) interna kompleksnost programske opreme – od uporabniškega programa do izvajanja strojne kode – je v primerjavi z ostalimi gradniki visokotehnoloških naprav velika. Programiranje je človekovo ustvarjalno delo, ki ga ni možno avtomatizirati. Vnos napak in/ali pomanjkljivosti je pri razvoju programske opreme dano dejstvo [2, str. 245-266]. Metode programskega inženirstva zmanjšajo

pomanjkljivosti in odpravijo napake, preden postanejo prepoznavne v delovanju sistema in/ali naprave.

Programsko opremo lahko delimo na uporabniške programe, opremo sistemov za krmiljenje v realnem času, distribuirane rešitve upravljanja proizvodnih procesov, sistemsko programsko opremo, programe informacijsko-komunikacijske infrastrukture, distribuirane poslovne aplikacije in drugo. Zakonitosti za uspešen razvoj teh različnih sistemov so skupne.

Proces izdelave programske opreme razdelimo v posamezne faze: definicija problema, študija izvedljivosti, analiza in določitev zahtev, načrtovanje sistema, strukturiranje programske rešitve, kodiranje, testiranja in verifikacije, sistemska integracija, testiranje celote, predaja produkta, vzdrževanje in posodabljanje programskega produkta tekom življenjske dobe.

Prednosti formalno strukturiranega in izvajanega razvoja programskih produktov, torej udejanjanja metod programskega inženirstva, pred ad hoc rešitvami so: večja produktivnost, projekt je bolj obvladljiv - posledično je delo manj stresno, zanesljivost delovanja programskega izdelka je bistveno večja; ni nujno, da čas in cena izdelave programskega izdelka bistveno odstopata od napovedi; čas učenja novih programerjev je krajši, kakovost znanja pa je višja.

Razvoj programske opreme po metodah programskega inženirstva sistemsko izloča razloge za neuspeh programskih izdelkov. Običajno so to napake v delovanju, do katerih prihaja skoraj naključno oziroma je kakovost programske rešitve slaba, preseženi so roki in stroški, nerazumljene in/ali nezadovoljene so uporabnikove potrebe, vzdrževanje je težavno. Do teh usodnih razlogov za neuspeh sicer pride zaradi neustreznega planiranja razvoja, nezadostno določenih ciljev projekta, prepovršnega razumevanja uporabnikovih potreb, nezadostnega sprotnega in končnega preverjanja funkcionalnosti, nezadostnega znanja razvojnikov in nezadostnega razumevanja potrebnih razvojnih vložkov razvijalca in uporabnika [3, str. 175–189].

Po metodah programskega inženirstva nameravano funkcionalnost programske opreme strukturiramo v diagrame, sestavljene iz hierarhično oblikovanih medsebojno povezanih funkcijskih blokov. Ti diagrami so formalizacija delovanja programske opreme.

Programsko opremo kodiramo v različnih programskih jezikih. Vsaka programska oprema se izvaja na nekem računalniku, v katerem nek mikroprocesor izvaja programske ukaze. Najbolj elementarno je programiranje v strojnem jeziku, ki je na nivoju ničel in enojk, nosilec informacije v digitalnem sistemu. Produktivnost takšnega programiranja je za izdelavo uporabniških programov neuporabno nizka. Vendar, tak je bil začetek programiranja (orodja za drugačno programiranje je bilo šele treba narediti). Nadgradnja strojnega jezika je zbirnik (assembler), ki že ima besedne programske ukaze. Ti ukazi so določeni z ozirom na arhitekturo mikroprocesorja – gre za programske ukaze na nivoju dela z registri, spominskimi lokacijami, logičnimi in aritmetičnimi enotami. Vsak mikroprocesor ima praktično svoj zbirnik. Ti programi niso prenosljivi. Na prvi pogled to ni pomembno, dejansko pa je prenosljivost programske opreme med njenimi najpomembnejšimi lastnostmi. Tudi takšno programiranje je za izdelavo uporabniških programov predolgotrajno in s tem neproduktivno. Kasneje, za nizkonivojskima strojnem jezikom in zbirnikom, so nastali višjenivojski programski jeziki, ki imajo sintakso in semantiko prilagojeno logičnim potrebam pisanja programov, ne pa arhitekturi mikroprocesorja.

Tehnološko so glavni del teh jezikov prevajalniki nizov programskih ukazov v zbirnik in naprej v strojno kodo mikroprocesorja. Uporabniško pa je glavni del teh jezikov nabor učinkovitih ukazov za programiranje, oziroma za »izražanje v programskem jeziku«. Ogromna je razlika med kodiranjem z ukazi, ki so »pisani na kožo« danemu mikroprocesorju in med kodiranjem z ukazi, ki so »pisani na kožo« uporabniku programerju oziroma so definirani kot gradniki za učinkovito pisanje uporabniških programov.

Prevajalnik iz zbornika v strojni jezik je bistveno lažje narediti kot prevajalnik iz višjih programskih jezikov (na primer C, pascal, BASIC)

v strojni jezik. Zato so zbirniki za različne mikroračunalnike lahko nastajali hkrati z razvojem mikroračunalnikov od šestdesetih let prejšnjega stoletja naprej. Na začetku računalniških tehnologij so v zbirniku (ker drugega ni bilo) pisali celo operacijske sisteme in uporabniške programe. Na primer, IBM PC DOS in Lotus 123 sta napisana v zbirniku.

Šele v sedemdesetih letih prejšnjega stoletja so nastali prevajalniki iz višjih programskih jezikov v zbirnik in/ali v strojno kodo za vsaj najbolj razširjene mikroprocesorske arhitekture. Takrat je nastajal programski jezik C (med 1969 in 1973), ki ga tudi danes najbolj uporabljamo za razvoj programske opreme vgradnih sistemov. Pred C-jem so nastali Fortran (1953), ALGOL (ALGO^rithmic Language), Simula, BASIC, pascal (1968) in objektni pascal. C-ju sledijo Visual BASIC, LabVIEW, objektni C, C++, C#, .NET, IEC 61131-3 ST (Strukturirani Tekst, *angl. structured text*), Java in PHP. Evolucija programskih jezikov je zvezna. Sedanji poudarek je na platformski neodvisnosti programskih rešitev.

Programski jezik C je nastal z namenom univerzalne uporabnosti (visoko in nizkonivojski jezik) v AT&T Bell Laboratories. Avtorja sta Dennis Ritchie in Brian Kernighan. Prve C prevajalnike so napisali za platforme z UNIX operacijskim sistemom. Kasnejši C prevajalniki za MS-DOS so omogočili masovno uporabo C jezika.

Programski jezik C vsebuje ukaze in podatkovne strukture, ki so namenjeni preglednemu in strukturiranemu pisanju uporabniških programov oziroma visokonivojskemu programiranju ter tudi ukaze in podatkovne strukture, ki so namenjeni programiranju na nivoju spominskih lokacij in posameznih bitov oziroma nizkonivojskemu programiranju. V C-ju so na primer napisani orodji Mathematica in MATLAB ter operacijski sistem UNIX. Na osnovi C-ja so razviti objektno orientirani jeziki C++, objektni C, C# in Java. Programsko sintakso in semantiko po C-ju povzema IEC 61131-3 ST za programiranje programabilnih logičnih krmilnikov (PLK-jev). Zaradi dobre logične zasnove in posledične uporabnosti C-ja so razvijalci napisali prevajalnike iz C kode v strojno kodo za večino

mikroprocesorjev. C kodo, ki je pisana po standardu ANSI (*angl. American National Standards Institute*), lahko prevajamo za izvajanje na različnih strojnih platformah in na različnih operacijskih sistemih ob zelo malo spremembah kode. Posledično je večina programske opreme za upravljanje visokotehnoloških naprav in procesov, implementiranih v vgradnih sistemih (*angl. embedded systems*), pisana v jeziku C. Leta 1990 je bil publiciran prvi ANSI C standard. Zadnji ANSI C standard je iz leta 2011.

Jezik C zaradi preišljene strukture ukazov lahko uporabljamo za pisanje obsežnih distribuiranih programov, samostojne programe, zaradi podpore delu s kazalci (*angl. pointers*) za direktno programiranje strojne opreme, pisanje operacijskih sistemov, skratka C se je zaradi njegove univerzalnosti smiselno naučiti. C podpira veliko večino konceptov programiranja. Za objektno programiranje je C nadgrajen v objektni C in v C++.

Pri razvoju C-ja je bilo pomembno, naj prevajalnik generira čim bolj efektivno strojno kodo (malo kode, nič odvečnosti, hitro izvajanje), naj omogoča direkten dostop do spomina oziroma do naslovov spominskih lokacij, in naj deluje s čim manj podpornimi knjižnicami. Posledično C-jeva zasnova omogoča pisanje programske opreme tudi na področjih, kjer je bil prej potreben zbirnik. Višjenivojski C je nadgradnja zmožnosti nizkonivojskega programiranja in omogoča univerzalno programiranje oziroma neodvisnost od strojne platforme.

Inženirji strojništva razvijajo sisteme za krmiljenje in spremljanje proizvodnje, integracijo proizvodnih s poslovnimi sistemi, krmiljenje in spremljanje distribuiranih proizvodnih in logističnih procesov, procesno avtomatizacijo in nove mehatronske naprave. Komponente visokotehnološke mehatronske naprave so mehanski sistem, elektronska platforma z mikrokrmilnikom za zajem podatkov, za izvajanje krmilnih algoritmov in krmiljenje aktuatorjev ter programska oprema. Mikrokrmilnike cenovno optimiramo – v napravo vgradimo samo toliko »računalniške moči«, kolikor je za delovanje naprave potrebno. Programsko opremo običajno razvijamo na zmogljivi delovni postaji in

jo v bolj ali manj narejenem stanju prenesemo na vgradni sistem naprave, kjer jo razvijemo do konca. Prenosljivost programske opreme z zmogljive na manj zmogljivo platformo je kar nujna za produktiven in kakovosten razvoj programske opreme za vgradne sistema. Tu pride do izraza v ANSI C vgrajena prenosljivost. Poleg tega, da omogočajo visoko in nizkonivojsko programiranje, so ANSI C programi teoretično popolnoma, praktično pa zelo prenosljivi med razvojnimi platformami in vgradnimi sistemi. Programiranje v ANSI C-ju omogoča tudi vzdrževanje vgradnih sistemov. Ko je iz različnih razlogov treba zamenjati mikrokontroler z novejšim (dobavljivost ali s časom potreba po razširitvi funkcionalnosti), je velika večina ANSI C kode v nespremenjeni obliki primerna za ponovno uporabo v novem sistemu.

Učiti se večino aktualnih programskih jezikov na zalogo predstavlja neučinkovito rabo energije in časa. Metodično se seznaniti z uporabo večine ukazov standardiziranega platformsko prenosljivega visoko in nizkonivojskega jezika, ki predstavlja osnovo tudi za druge moderne jezike, pa je smiselna osnova za pisanje distribuiranih programskih orodij in za mehatronsko interdisciplinarno delo.

Komponenta mehatronskega interdisciplinarnega projekta je vgradni sistem oziroma sodobna implementacija krmiljenja.

1.2 Vgradni sistem

Vgradni sistem je krmilni sistem, ki sestoji iz mikrokontrolerja (*angl. microcontroller*), vseh potrebnih perifernih enot – zaznaval, aktuatorjev, torej iz celotne elektronike in ustrezne programske opreme, ki določa funkcionalnost visokotehnološke naprave [4, str. 5-31].

Še pred desetimi leti je večina naprav delovala brez vgrajenih mikroročunalnikov. Krmilna logika je bila izvedena na nivoju logičnih vrat in spominskih celic ali celo z zobniki, kontakti in vzmetmi (kot so še vedno narejeni na primer krmilniki cenejših pralnih in pomivalnih strojev). V teh tehnologijah praktično ni možno izvesti kompleksnejših upravljaljskih funkcij. Od iznajdbe računalnika naprej pa je zmogljivejši krmilni sistem realiziran z ločenim računalnikom, povezanim z napravo

ali sistemom z množico kablov za zajemanje podatkov in za krmiljenje aktuatorjev.

Vgradni sistem je del same naprave. Vgrajeno mikroračunalniško krmiljenje naprave poenostavi sistemski razvoj in omogoča fleksibilnost delovanja naprave – spremembe in izboljšave so narejene v programski opremi. Visokotehnološkega mehatronskega izdelka se danes brez vgradnega sistema praktično ne da več narediti. Uporabnik pogosto niti ne ve, da je naprava računalniško krmiljena. Z uporabniškega stališča način krmiljenja niti ni pomemben. Obstoj vgradnih sistemov je spremenil celoten koncept delovanja in uporabniške izkušnje visokotehnoloških naprav. Za ponazoritev navajamo dva primera:

Pri ščetki na sliki 1 mikrokrmilnik krmili intenzivnost in čas trajanja vibracij ščetke, brezkontaktno indukcijsko polnjenje ter prikaz stanja baterije. Nekoč je bila zobna ščetka sestavljena samo iz ročaja in ščetin. Razlika med staro in novo zobno ščetko je očitna in velika. Čiščenje na novi način je uspešnejše (odstranjevanje oblog z vibracijami) in manj uničujoče kot prej (manj je drgnjenja in posledično obrabe).

Pri parnem kotlu na sliki 1 zmogljivejši mikrokrmilnik krmili več deset kilovatov električne moči za izvajanje termičnih procesov kuhanja ali vrenja ali poparjevanja do sto petdeset litrov hrane. Hkrati krmili tudi uporabniški vmesnik na dotik, prikaz spremenljivk, beleženje procesnih spremenljivk in podatkovno/servisno komunikacijo prek internetnega omrežja.

Konceptualno sta si krmilna sistema obeh naprav zelo podobna. Seveda kompleksnost izvedbe narašča z zahtevnostjo krmiljenja.

Mikrokrmilnik krmili tudi moderno računalniško miško, vsak podatkovni disk, tiskalnik, avtomat za hrano in/ali pijačo, telefon, multimedijske naprave, veliko igrač in predvsem tehnološko zahtevne produkte, kjer zmogljivo krmiljenje res pride do izraza: kompleksna industrijska krmiljenja, sistemi za raziskovanje vesolja, krmilni sklopi letal in sklopi zemeljskih in vodnih transportnih sredstev, merilne naprave in sistemi, industrijski roboti, pametne zgradbe, hišna avtomatizacija, mikrovalovne

pečice, razne gospodinjske naprave, medicinske diagnostične naprave, roboti za medicinske operacije, prodajni terminali in mnogo drugega.



Slika 1: Z vgradnim sistemom krmiljena zobna ščetka Sonicare® in Parni kotel Kogast®

Trenutno na vsak osebni računalnik izdelajo več kot sto vgradnih sistemov. Razvoj krmiljenja naprav z vgradnimi sistemi je omogočil krajše čase razvoja novih naprav in več funkcionalnosti ob dosegljivi ceni. Nove naprave za industrijsko in osebno rabo so informacijsko povezljive in imajo intuitivne uporabniške vmesnike. Predvsem v krmiljenju procesov so na razpolago nove možnosti pri zaznavalnih, krmiljenju samem in pri aktuatorjih. Potencialen rezultat so naprave s

popolnejšim krmiljenjem, posledično z večjim izkoristkom, torej z manjšo porabo energije in z manj škodljivim vplivom na okolje.

Vgradni sistemi sestojijo iz strojne in programske opreme.

1.2.1 Strojna oprema vgradnega sistema

Strojna oprema je danes izključno v domeni elektrotehnike in elektronike. Prve implementacije krmiljenja procesov so bile izvedene z mehanskimi in/ali elektromehanskimi sistemi in še to le takrat, kadar je bilo to res nujno za implementacijo delovanja in/ali določeno tehnološko prednost. Takšno uporabo so predstavljali elektromehanski računalniki za določanje parametrov streljanja na vojaških ladjah in sklopi krmilnih sistemov na vojaških letalih za podporo in razbremenitev pilota v drugi svetovni vojni.

Z razvojem polprevodniških tehnologij so krmilne naprave postale manjše in zmogljivejše. Razvoj digitalne tehnologije je najprej omogočil industrijsko avtomatizacijo z logičnimi vezji. Njihovi gradniki so logična vrata *AND*, *OR*, *NAND*, *NOR*, *XOR*, *XNOR* in spominski elementi *D flip flopi* ter *SR flip flopi*. Na tak način lahko realiziramo logične sekvenčne avtomate, gradnja kompleksnejše avtomatizacije pa ni ekonomsko upravičena.

Leta 1971 je podjetje Texas Instruments vložilo patentno prijavo za prvi mikroprocesor narejen v enem samem vezju (*angl. single-chip microprocessor*). Osemdeseta leta prejšnjega stoletja zaznamuje začetek tehnologije vgradnih sistemov.

Gradniki današnjih vgradnih sistemov so mikrokrmilniki, potencialno vsa kataloška integrirana vezja in drugi elektronski elementi. Kadar zahteve implementacije tehnološkega procesa to zahtevajo in je ekonomsko upravičeno, izdelamo tudi dodatna integrirana vezja po naročilu (*angl. custom integrated circuit*). Enostavnejša tehnologija, ki to omogoča, je FPGA (*angl. Field Programmable Gate Array*). Tu gre za programiranje električnih povezav v vnaprej matrično oblikovanem vezju elektronskih gradnikov. Dražji in popolnoma optimiran način implementacije je

izdelava ASIC (*angl. Application Specific Integrated Circuit*). Samo velike serije izdelka in visoke tehnološke zahteve upravičijo tak razvojni strošek.

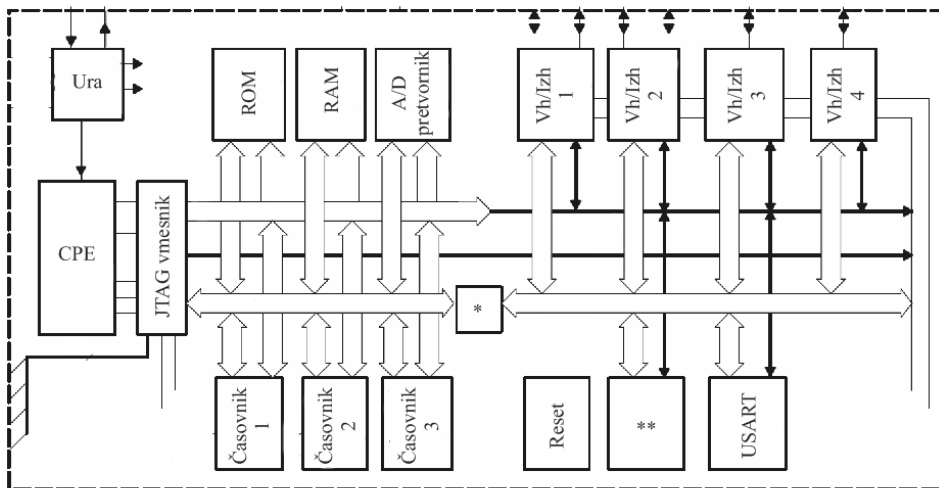
Vsak mikrokrmilnik vsebuje mikroprocesor. Na splošno le-ta sestoji iz aritmetične logične enote (*angl. arithmetic logic unit ALU*), instrukcijskega dekoderja, polja registrov (spominskih lokacij) in kontrolne enote.

V najosnovnejšem smislu je mikroprocesor naprava za izvajanje algebre. Prvi izdelki z vgrajenim mikroprocesorjem so bili ročni kalkulatorji, ki vsakršno računanje močno poenostavijo. Pred tem so si inženirji pri računanju pomagali s pomičnim mehanskim računalom (*angl. slide rule, nem. Rechenschieber*), ki ga je kalkulator povsem izpodrinil. Ob primerljivi ceni omogoča bistveno več in je enostavnejši za uporabo.

Naprava, ki izvaja algebro, izvaja tudi logične funkcije. Le-te so definirane v dvojiškem številskem sistemu, ki je za računalniško implementacijo bistveno primernejši od desetiškega, ki ga uporabljamo za vsakodnevno računanje.

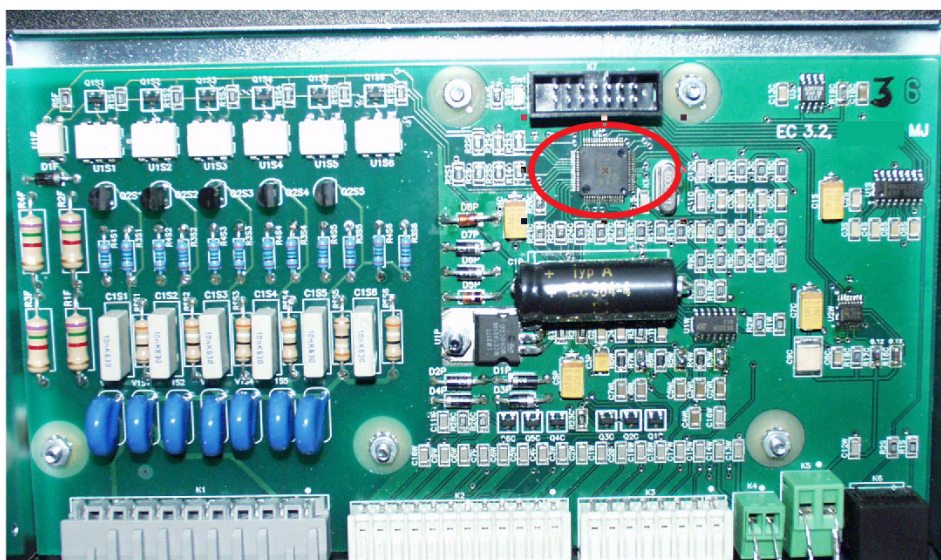
Za rabo mikroprocesorja v krmiljenju procesov v realnem času je bilo treba dodati bistvene periferne enote za zajem podatkov krmiljenega sistema, krmiljenje aktuatorjev in prenos podatkov po komunikacijskem omrežju. Takšno integrirano napravo imenujemo mikrokrmilnik.

Mikrokrmilnik z blokovno shemo na sliki 2 vsebuje mikroprocesor in periferijo za krmiljenje industrijskih procesov in naprav v realnem času: programabilne vhodno-izhodne enote, ki jih lahko precej poljubno programsko nastavimo kot digitalne in analogne podatkovne vhode ter izhode, primerjalnike signalov, časovnike za merjenje časov in zakasnitev, več podatkovnih vhodov s sposobnostjo proženja prekinitve osnovnega delovanja in takojšnje obdelave logičnega pogoja v trenutku, ko se zgodi. Dodanih je še več analogno digitalnih pretvornikov za zajem analognih veličin in enota za komuniciranje z mrežnim informacijskim sistemom.



Legenda: * vmesnik med 16 in 8 bitnim podatkovnim vodilom
 ** primerjalnik

Slika 2: Blokovna shema mikrokrmilnika družine MSP430, Texas Instruments



Slika 3: Mikrokrmilnik s periferijo v napravi Golden Egg®

Mikrokrmilnik na sliki 3 je obkrožen z rdečo elipso in geometrijsko predstavlja le majhen del krmilnega sistema. V tem konkretnem

mikrokrmilniku je shranjenih in se izvaja približno pet tisoč vrstic kode C. Ostali elementi na sliki 3 so močnostna stikala in signalni filtri ter napetostne zaščite vhodnih signalov. Ti elementi zahtevajo določeno površino oziroma volumen, saj preklapljujejo konkretne moči pri omrežni napetosti. Popolna miniaturizacija tiskanega vezja ni možna zaradi omrežne napetosti med posameznimi povezavami vezja. S podobno grajenimi sistemi krmilimo različne industrijske procese, naprave in sklope naprav.

Vmesna stopnja med delovno postajo (*angl. workstation*), uporabljeno kot računalnik za upravljanje industrijskega procesa in/ali naprave ter med vgradnim sistemom za upravljanje procesa in/ali naprave predstavlja Programabilni Logični Krmilnik - PLK (*angl. Programmable Logic Controller – PLC*), slika 4.



Slika 4: Programabilni modularno grajen logični krmilnik Beckhoff®

PLK-ji so namenjeni krmiljenju industrijskih procesov in splošni industrijski avtomatizaciji. PLK-je programiramo v lestvičastih diagramih (*angl. ladder diagram, LD*), v diagramih funkcijskih blokov (*angl. function block diagram, FBD*), v sekvenčnih funkcijskih diagramih (*angl. sequential function chart, SFC*), v zaporedjih instrukcij (*angl.*

instruction list. IL), v strukturiranem tekstu (*angl. structured text, ST*, zelo podobno C-ju), lahko pa tudi v C-ju [5, str. 40] (določila PLK programskega standarda IEC 61131-3). PLK-ji so narejeni modularno. Za vsako aplikacijo sestavimo potrebno konfiguracijo iz posameznih tipskih gradnikov. PLK-ji kot vgradni sistemi niso primerni – še vedno so preveč univerzalni, oziroma ne dovolj specializirani za krmiljenje serijsko izdelanih naprav. Cena in fizični volumen takih krmilnih naprav bi bila prevelika! Le optimizacija mikroračunalnika na nivoju integriranih vezij omogoča največjo fleksibilnost ter zmogljivostno in cenovno optimizacijo serijsko izdelane visokotehnološke moderne naprave.

Uporabniški vmesniki vgradnega sistema so lahko zelo različni. Vmesniki polpreteklega časa so narejeni s folijskimi tipkovnicami in s prikazovalniki s svetlečimi diodami. Najmodernejši vmesniki imajo zaslone LCD (*angl. Liquid Crystal Display*), ki so občutljivi na dotik. Takšne vmesnike imajo na primer pametni telefoni. Narejeni so v milijonskih serijah, zato so lahko in morajo biti popolnoma funkcionalno optimizirani. Nudijo najboljšo možno uporabniško izkušnjo, ki si jo uporabnik lahko želi pri vseh visokotehnoloških napravah. Veliko proizvodov pa ne prenese dodatne cene vseh tehnologij, ki so uporabljene v pametnem telefonu. Mnogo naprav je narejenih v majhnih serijah, ker so namenjene specifični uporabi in ne vsakemu prebivalcu globalnega sveta. Količina ekonomsko upravičenega razvojnega dela je tesno odvisna od velikosti serije naprave.

1.2.2 Programska oprema vgradnega sistema

Mikrokrmilnik je namensko konstruiran za krmiljenje procesov in naprav v realnem času. Za konkretno krmiljenje je treba programsko nastaviti delovanje vseh perifernih enot in napisati programsko opremo, ki določa funkcionalnost naprave v realnem času – brez nenačrtovanih zakasnitev delovanja in z zanesljivim delovanjem ob vsakršnih kombinacijah ter zaporedjih vhodnih podatkov.

Današnji de facto standard za razvoj vgradne programske opreme je ANSI C, oziroma standardiziran jezik C. Kodo C, ki je pisana po

standardu ANSI, lahko prevajamo za različne strojne platforme in za različne operacijske sisteme, ob relativno majhnih spremembah kode.

1.3 Učenje jezika C

Referenčni dokument jezika C je [6] v slovenskem prevodu [7]. Sintaksa in semantika programskih ukazov C sta podani na primer v [6, str. 191–239], [8, str. 905–1004] in [9, str. 429–466].

Funkcionalnost standardne knjižnice C je podana na primer v [6, str. 241–258], [9, str. 467–492] in [10].

Učenje programskega jezika strukturiramo za postopno dodajanje kompleksnosti kot na primer v [11]. Tudi v tem učnem pripomočku so naloge in koda njihovih rešitev strukturirani v zaključene celote. Razpored poglavij je povzet po [9]. Kompleksnost nalog narašča postopoma, saj v naslednjih poglavjih uporabljamo znanje iz prejšnjih poglavij. C je jezik s širokim krogom uporabnikov, kratke primere najdemo tudi na medmrežju, kot na primer v [12]. Znanje C-ja nam predstavlja programersko osnovo, ki jo nadgrajujemo s posebnostmi programiranja za izdelavo krmiljenja v realnem času in z drugimi proceduralnimi in objektno orientiranimi jeziki za izdelavo distribuiranih programskih rešitev.

2 Prvi programi – prevajanje in izpis

Poudarki:

- Prevajanje programa v izbranem programskem okolju.
- Vključitev knjižnice *stdio* (standardni I/O) v program.
- Argumenti programa *main()*.
- Znaki za vključitev komentarja.
- Izpis nizov s *printf()*.
- Kontrolni znak ‘\n’ v izpisu.
- Kontrolni znak za oblikovanje podatka v izpisu – ‘%d’ – decimal, desetiški izpis celega števila.
- Vrnjena vrednost funkcije *printf()*.
- Izpis celoštevilskega števila s *printf()*.
- *#pragma* ukazi prevajalniku.

Opombi:

- V navodilih nalog so šumniki pravilno pisani. V zahtevkih po izpisih programov in v rešitvah nalog pa šumnikov ni. Velika večina programskih orodij za ANSI C programiranje za zapis črk uporablja le standardno tabelo 256 ASCII znakov brez šumnikov.
- Za ANSI C prevajalnik s piko ločimo celi in decimalni del realnega števila. Oba dela nimata dodatnih znakov zaboljšanje preglednosti (na primer knjižno oblikovan zapis 1.2345,67 tu pišemo kot 12345.67).

2.1 Naloge

2.1.1 N: Dober dan, svet

Program naj izpiše:

“Dober dan, svet!”

“Hello world!”

Namen: Naš prvi program. Nepisano pravilo je, da je v vsakem jeziku prvi izdelek napis “Hello world“, torej “Dober dan, svet!“

Izpis:

```
Pozdravljen svet!  
Hello world!
```

//-----

2.1.2 N: Navodilo programerju

Program naj izpiše:

1. V C jeziku je pomembno razlikovati med velikimi in malimi črkami.
2. Program začnemo s funkcijo `main()`.
3. Vsako funkcijo začnemo z zavitim oklepajem in končamo z zavitim zaklepajem.
4. Vsak programski stavek zaključimo s podpicjem.“

Namen: Raba funkcije `printf()` s kontrolnim znakom za novo vrsto in z zapisom oklepaja.

Izpis:

```
1. V C jeziku je pomembno razlikovati med velikimi in malimi črkami.  
2. Program začnemo s funkcijo main().  
3. Vsako funkcijo začnemo z oklepajem in končamo z oklepajem.  
4. Vsak programski stavek zaključimo s podpicjem.
```

//-----

2.1.3 N: *printf()*, vrnjena vrednost

S funkcijo *printf()* izpišimo »Zdaj je četrtek zvečer«. V naslednjem *printf()* izpišimo vrnjeno vrednost prvega *printf()*, - dolžino prvega izpisa.

Namen: Izpis vrnjene vrednosti funkcije *printf()*. Funkcije pišemo tako, da z vrnjeno vrednostjo informirajo o načinu izvajanja ali o vrsti napake v izvajanju funkcije.

Izpis:

```
Zdaj je četrtek zvečer.  
Urnjena vrednost funkcije printf() je: 25
```

//-----

2.1.4 N: Osebna vizitka

Napišimo program, ki s funkcijo *printf()* semi-grafično izpiše vizitko poljubne osebe v poljubni obliki.

Namen: Izpis besedila z ambicijo grafičnega oblikovanja.

Izpis:

```
*****  
*                               *  
*   Janez Franc Novakov         *  
*                               *  
*   Cesta v mestni log 42      *  
*   1000 Ljubljana            *  
*                               *  
*   +386 41 555 000           *  
*                               *  
*****
```

//-----

2.2 Rešitve (R)

2.2.1 R: Dober dan, svet

/* najprej pojasnilo o #pragma – pragmatic – pragmaticen – praktičen: vsi ukazi, ki sledijo po #pragma do konca vrste, so namenjeni prevajalniku iz C kode v strojni jezik.

#pragma hdrstop – header stop: vsa koda pred vrstico #pragma hdrstop se prevede ob zahtevku po prevajanju le, če je v tej kodi kaksna sprememba, sicer pa se le doda zadnji prevod te kode trenutnemu prevodu ostanka kode za vrstico #pragma hdrstop. Namen: hitrejše prevajanje, kadar je na začetku datoteke veliko #include stavkov z deklaracijami in/ali definicijami podatkovnih struktur in funkcij iz knjižnic in/ali preostalnih datotek, gradnikov konkretnega programa.

#pragma argsused – arguments used: velja za argumente funkcije neposredno za #pragma argsused. Prevajalnik naj ne opozori, kadar argumenti funkcije v funkciji niso uporabljeni. main() običajno klicemo brez argumentov in ne potrebujemo opozorila na to pri vsakem prevajanju.

```
*/  
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    printf("\nPozdravljen svet!\n");  
    printf("Hello world!\n");  
  
    return 0;  
}  
  
//-----
```

2.2.2 R: Navodilo programerju

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    printf ("\n1. V C jeziku je pomembno razlikovati med velikimi in malimi  
crkami.\n");  
    printf ("2. Program zacnemo s funkcijo main\(\).\n");  
    printf ("3. Vsako funkcijo zacnemo z oklepajem in koncamo z oklepajem.\n");  
    printf ("4. Vsak programski stavek zakljucimo s podpicjem.\n");  
  
    printf("\n");  
  
    return 0;  
}  
//-----
```

2.2.3 R: *printf()*, vrnjena vrednost

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int x;  
    x = printf("\nZdaj je cetrtek zvecer.\n");  
  
    printf("Vrnjena vrednost funkcije printf(\n) je: %d\n", x);  
  
    return 0;  
}  
//-----
```

2.2.4 R: Osebna vizitka

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    printf("\n*****\n");  
    printf("                *\n");  
    printf("    Janez Franc Novakov    *\n");  
    printf("                *\n");  
    printf("    Cesta v mestni log 42    *\n");  
    printf("        1000 Ljubljana      *\n");  
    printf("                *\n");  
    printf("        +386 41 555 000     *\n");  
    printf("                *\n");  
    printf("*****\n");  
  
    return 0;  
}  
//-----
```


3 Spremenljivke, podatkovni tipi in aritmetični izrazi

Poudarki:

- Deklaracija spremenljivk tipa *int*, *float*, *double*, *long double*, *char*.
- Niz spremenljivk tipa *char*.
- Aritmetični operatorji $+$, $-$, $*$, $/$, $\%$ s celoštevilskimi spremenljivkami tipa *int*.
- Aritmetični operatorji $+$, $-$, $*$, $/$ s tipi spremenljivk za realna števila.
- Kompaktni aritmetični operator $+=$.
- Računske operacije med argumenti tipov *int* in *float*.
- Potenciranje x^y , *pow(x,y)*.
- Uporaba in oblike izpisov velikih in majhnih števil z eksponenti, $n * 10^m$, *nEm*.
- Vnos celih in realnih števil s *scanf()*.
- Izpis niza s *puts()*.
- Izpis spremenljivk tipa *int*, *float*, *double*, *long double*, *char* s *printf()*.
- Argumenti vseh funkcij iz knjižnice *math* so tipa *double*.

3.1 Naloge

3.1.1 N: *scanf()*, *printf()*

Napišimo program, ki naj z uporabo *printf()*, *scanf()* izvede:

Vpisite avtomobilsko znamko in model: npr. Renault

Vpisite model avtomobila: npr. Clio

Vpisite ceno: npr. 1400

Avtomobil Renault Clio stane 1400 Eur.

Namen: Vpis podatka s *scanf()*, sestavljen izpis.

Izpis:

```
Upisite avtomobilsko znamko: Skoda
Upisite model avtomobila: Fabia
Upisite ceno: 9999
Avtomobil Skoda Fabia stane 9999 Eur.
```

//-----

3.1.2 N: Kalkulator

Napišimo program, ki prebere dve celi števili ter izpiše vsoto, razliko, produkt, kvocient in ostanek pri deljenju. Delovanje:

Vpisite 1. celo število: npr. 5

Vpisite 2. celo število: npr. 4

$$6 + 4 = 10$$

$$6 - 4 = 2$$

$$6 * 4 = 24$$

$$6 / 4 = 1$$

$$6 \% 4 = 2$$

Namen: Delo s celimi števili: vnos, uporaba operatorjev algebre, izpis.

Izpis:

```
Po vpisu dveh celih števil dobite vsoto, razliko, produkt, kolicnik in ostanek.
Vpisite 1. celo število: 45
Vpisite 2. celo število: 62
45 + 62 = 107
45 - 62 = -17
45 * 62 = 2790
45 / 62 = 0
45 % 62 = 45
```

//-----

3.1.3 N: Polinom

Napišimo program, ki bo izračunal $3 * x^3 - 5 * x^2 + 6$ za $x = 2.55$

Namen: Računanje z realnimi števili, spremenljivke tipa *float*, funkcija *pow(x,y)*.

Možen izpis:

```
Racunamo 3 * x**3 - 5 * x**2 + 6 za x=2.55.
Rezultat je 23.231623.
Razlika med mnozenjem in rabo pow(x,y) je 0.000000.
```

//-----

3.1.4 N: Ulomek z eksponenti

Napišimo program, ki bo izračunal in izpisal rezultat za izraz $(3.31 * 10^{-8} * 2.01 * 10^{-7}) / (7.16 * 10^{-6} + 2.01 * 10^{-8})$

Namen: Računanje z majhnimi (in velikimi) števili, spremenljivke tipa *float*, *double*, *long double*, rezultat izpisan v različnih formatih.

Možen izpis:

```
Racunamo <3.31E-8 * 2.01E-7> / <7.16E-6 + 2.01E-8>
Izpis z %i, razumljivo narobe:
Rezultat je -1073741824
Izpis z %e:
Rezultat je 9.266027e-10
Izpis z %E, USTREZNO:
Rezultat je 9.266027E-10
Izpis z %f:
Rezultat je 0.000000
Izpis z %g, je krajse izmed %e in %f:
Rezultat je 9.26603e-10
Izpis z %G, je krajse izmed %E in %f, NAJBOLJ USTREZNO
Rezultat je 9.26603E-10
```

//-----

3.1.5 N: Obseg in površina kroga

Program naj izračuna obseg in površino kroga in naj podatke izpiše na 3 decimalna mesta natančno.

Namen: Določitev π , izpis realnih števil na n decimalnih mest.

Izpis:

```
Racunamo obseg in površino kroga, vpišite polmer: 32.675
Radij kroga je 32.675, obseg kroga je 205.303, površina kroga je 3354.139.
```

//-----

3.1.6 N: Vsota kotov

Napišimo program, ki prebere dva kota v stopinjah, minutah in sekundah (celoštevilske vrednosti). Nato naj izračuna in izpiše njuno vsoto. Rezultat naj bo spet v stopinjah, minutah in sekundah.

Namen: Seštevanje in pretvarjanje podatkov v celih številih. Za doseg tega cilja je smiselno računati v najmanjših enotah (v tem primeru v sekundah) in rezultat ob izpisu prilagoditi v željeno obliko.

Izpis:

```
Racunamo vsoto dveh kotov:
Upisite prvi kot:
    stopinje: 12
    minute: 55
    sekunde: 59
Upisite drugi kot:
    stopinje: 10
    minute: 15
    sekunde: 6
Vsota obeh kotov je 23 stopinj, 11 minut in 5 sekund.
```

//-----

3.1.7 N: Preračun iz °F v °C

Program naj preračuna Fahrenheitove v Celzijeve stopinje.

Namen: Vnos celega števila, izpis realnega števila, algebra s celim in realnim številom.

Izpis:

```
Celostevilske F stopinje, pretvorba v C stopinje:
90
90 stopinj Fahrenheita je 32.222221 stopinj Celzija
```

//-----

3.1.8 N: Pretvorba iz radianov v stopinje

Napišimo program, ki prebere geometrijski kot kot realno število v radianih in ga izpiše kot realno število v stopinjah.

Namen: Določitev števila π , vpis in izpis ter račun z realnimi števili.

Izpis:

```
Upisite kot v radianih: 3.14
3.140000 radianov = 179.908752 stopinj.
```

//-----

3.1.9 N: Obrat števk

Napišimo program, ki prebere štirimestno številko in jo izpiše v obratnem vrstnem redu.

Namen: Razstavljanje in sestavljanje številke v števke. Uporaba celoštevilskega deljenja in ostanka celoštevilskega deljenja.

Izpis:

```
Obracam o štirimestno številko.
Upisite štirimestno številko: 8742
Obrnjena številka 8742 je 2478.
```

//-----

3.2 Rešitve (R)

3.2.1 R: scanf(), printf()

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
char cZnamkaAvta[80], cModelAvta[80];  
int iCena;  
  
printf("\nVpisite avtomobilsko znamko: ");  
scanf("%s",&cZnamkaAvta);  
printf("Vpisite model avtomobila: ");  
scanf("%s",&cModelAvta);  
printf("Vpisite ceno: ");  
scanf("%d",&iCena);  
printf("Avtomobil %s %s stane %d Eur.\n",cZnamkaAvta, cModelAvta, iCena);  
return 0;  
}  
//-----
```

3.2.2 R: Kalkulator

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int iA, iB;  
    int vsota, razlika, produkt, kolicnik, ostanek;  
    puts("\nPo vpisu dveh celih stevil dobite vsoto, razliko. produkt, kolicnik in  
ostanek.");  
  
    printf("Vpisite 1. celo stevilo: ");  
    scanf("%d", &iA);  
    printf("Vpisite 2. celo stevilo: ");  
    scanf("%d", &iB);  
  
    vsota = iA + iB;  
    razlika = iA - iB;  
    produkt = iA * iB;  
    kolicnik = iA / iB;  
    ostanek = iA % iB;  
  
    printf("%d + %d = %d\n", iA, iB, vsota);  
    printf("%d - %d = %d\n", iA, iB, razlika);  
    printf("%d * %d = %d\n", iA, iB, produkt);  
    printf("%d / %d = %d\n", iA, iB, kolicnik);  
    printf("%d %% %d = %d\n", iA, iB, ostanek);  
    puts("");  
    return 0;  
}  
//-----
```


3.2.3 R: Polinom

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    float x = 2.55;  
  
    float y = 3 * x*x*x - 5 * x*x + 6;  
    float yy = 3 * pow(x,3) - 5 * pow(x,2) + 6;  
  
    float fRazlika = y - yy;  
  
    puts("\nRacunamo 3 * x**3 - 5 * x**2 + 6 za x=2.55.");  
    printf ("Rezultat je %f.\n", y);  
    printf ("Razlika med mnozenjem in rabo pow(x,y) je %f.\n", fRazlika);  
    puts("");  
  
    return 0;  
}  
//-----
```

3.2.4 R: Ulomek z eksponenti

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    // za prikaz rezultata uporabimo exponentni format  
    // (3.31 x 10-8 x 2.01 x 10-7) / (7.16 x 10-6 + 2.01 x 10-8)  
  
    // Mozna deklariranja za realno stevilo so:  
    // "float": 32 bitov, 3.4E-38 <-> 3.4E38  
    // "double": dvojni "float", 64 bitov  
    // opcijski modifier "long" poveca tocnost tipa "double", 80 bits  
  
    float fRezultat;  
  
    puts("\nRacunamo (3.31E-8 * 2.01E-7) / (7.16E-6 + 2.01E-8)\n");  
  
    fRezultat = (3.31E-8 * 2.01E-7) / (7.16E-6 + 2.01E-8);  
    //double dRezultat = (3.31E-8 * 2.01E-7) / (7.16E-6 + 2.01E-8);  
  
    puts ("Izpis z %i, razumljivo narobe:");  
    printf ("Rezultat je %i\n\n", fRezultat);  
    // %i: -1073741824 je narobe, kar je razumljivo: float je izpisan kot int  
  
    puts ("Izpis z %e:");  
    printf ("Rezultat je %e\n\n", fRezultat); // eXponent  
    // %e: 9.266027e-10  
  
    puts ("Izpis z %E, USTREZNO:");  
    printf ("Rezultat je %E\n\n", fRezultat); // Exponent
```

```
// %E: 9.266027E-10
```

```
puts ("Izpis z %f:");  
printf ("Rezultat je %f\n\n", fRezultat); // float  
// %f: 0.000000
```

```
puts ("Izpis z %g, je krajse izmed %e in %f:");  
printf ("Rezultat je %g\n\n", fRezultat);  
// %g: 9.26603e-10
```

```
puts ("Izpis z %G, je krajse izmed %E in %f, NAJBOLJ USTREZNO");  
printf ("Rezultat je %G\n\n", fRezultat);  
// %g: 9.26603E-10
```

```
return 0;  
}  
//-----
```

3.2.5 R: Obseg in površina kroga

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define PI 3.1416  
  
    double r, obseg, ploscina;  
  
    printf("\nRacunamo obseg in povrsino kroga, vpisite polmer: ");  
    scanf("%lf", &r);  
  
    obseg = 2 * PI * r;  
    ploscina = PI * pow(r,2);  
  
    printf("Radij kroga je %.3f, obseg kroga je %.3f, povrsina kroga je %.3f.\n\n", r,  
    obseg, ploscina);  
    return 0;  
}  
//-----
```

3.2.6 R: Vsota kotov

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int vsota, n;  
    int st, min, sek;  
  
    vsota = 0;  
    // smiselno je vse racunati v sekundah, da bo vse v celostevilskem racunanju  
  
    puts("\nRacunamo vsoto dveh kotov:");  
    puts("Vpisite prvi kot:");  
  
    // stopinja: 3600 sekund  
    printf("    stopinje: "); scanf("%d", &n); vsota += 3600*n;  
  
    // minuta: 60 sekund  
    printf("    minute: "); scanf("%d", &n); vsota += 60*n;  
  
    printf("    sekunde: "); scanf("%d", &n); vsota += n;  
  
    printf("Vpisite drugi kot:\n");  
    printf("    stopinje: "); scanf("%d", &n); vsota += 3600*n;  
    printf("    minute: "); scanf("%d", &n); vsota += 60*n;  
    printf("    sekunde: "); scanf("%d", &n); vsota += n;  
  
    // vsota je v sekundah, operator % 60 da ostanek sekund  
    sek = vsota % 60;  
    // minut 60 krat manj od sekund, operator % 60 da ostanek minut  
    min = (vsota / 60) % 60;
```

```
st = vsota / 3600;    // celostevilsko deljenje s 3600
printf("Vsota obeh kotov je %i stopinj, %i minut in %i sekund.\n\n", st, min,
sek);

return 0;
}
//-----
```

3.2.7 R: Preračun iz °F v °C

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    // C = (F - 32) / 1.8  
  
    int iFStopinje;  
    float fCStopinje;  
  
    puts("\nCelostevilske F stopinje, pretvorba v C stopinje: ");  
    scanf("%d", &iFStopinje);  
  
    fCStopinje = (iFStopinje - 32) / 1.8;  
  
    printf ("%i stopinj Fahrenheita je %f stopinj Celzija\n", iFStopinje, fCStopinje);  
    return 0;  
}  
//-----
```

3.2.8 R: Pretvorba iz radianov v stopinje

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    const double PI = acos(-1);  
  
    float rad, st;  
    //double rad, st;  
  
    printf("\nVpisite kot v radianih: ");  
    scanf("%f", &rad);  
    //scanf("%lf", &rad);  
  
    st = rad * 180 / PI;  
  
    printf("%f radianov = %f stopinj.\n", rad, st);  
    //printf("%lf radianov = %lf stopinj.\n", rad, st);  
    return 0;  
}  
  
//-----
```


3.2.9 R: Obrat števk

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
  
    int iNTisocic, iNStotic, iNDesetic, iNEnic;  
    int iStevilka, iObrnjena;  
  
    puts("\nObracamo stirimestno stevilko.");  
    printf("Vpisite stirimestno stevilko: ");  
    scanf("%d", &iStevilka);  
  
    iNEnic = iStevilka % 10;  
    iNDesetic = (iStevilka / 10) % 10;  
    iNStotic = (iStevilka / 100) % 10;  
    iNTisocic = (iStevilka / 1000) % 10;  
    iObrnjena = 1000 * iNEnic + 100 * iNDesetic + 10 * iNStotic + iNTisocic;  
  
    printf("Obrnjena stevilka %d je %d.\n", iStevilka, iObrnjena);  
    // ali:  
    // printf("Obrat številke %d je %d%d%d%d.\n", iStevilka, iNEnic, iNDesetic,  
    // iNStotic, iNTisocic);  
  
    return 0;  
}  
  
//-----
```

4 Programske zanke

Poudarki:

- Zanka *for* (inicializacija; logični pogoj; sprememba) { stavki; };
- Zanka *do* { stavki; } *while* (logični pogoj);
- Zanka *while* (logični pogoj) { stavki; }.
- Logični pogoji za izvajanje zank: $a \leq b$, $c \neq n$, $n-- > 0$.
- Inkrementi za izvajanje zank: $++i$; $i++$; $i+=n$.
- Izhod iz zanke z *break*;
- V zanko vgnezdena zanka.
- Izpis ASCII znakov tipa *char* v zanki po različnih kriterijih izbire.
- Branje črk vrstice v zanki z *getchar()*, pisanje črk vrstice v zanki s *putchar()*.
- Argumenti krožnih funkcij iz knjižnice *math* so v radianih in so tipa *double*.
- Izpisi v zanki *for* (inicializacija; logični pogoj; inkrement) { *printf()*; };

4.1 Naloge

4.1.1 N: Izpis ASCII znakov, [0 .. 255]

Napišimo program, ki bo izpisal tabelo ASCII znakov od 0 do 255.

Namen: ASCII znaki, izpis v zanki.

Izpis:

```
0
1 @
2 @
3 ♥
4 ♦
5 ♣
6 ♠
7
8
9
10
11 ♂
12 ♀
242 ≥
243 ≤
244 ∫
245 ∫
246 ÷
247 ∞
248 ∞
249 .
250 .
251 √
252 n
253 z
254 ■
255
```

//-----

4.1.2 N: Izpis ASCII znakov iz intervala

Program naj izpiše ASCII znake iz intervala med dvema vnešenima številcama.

Namen: Izpis določenih ASCII znakov v zanki.

Izpis:

```
Upisite spodnjo mejo v ASCII tabeli: 48
Upisite zgornjo mejo v ASCII tabeli: 57
48 = '0'
49 = '1'
50 = '2'
51 = '3'
52 = '4'
53 = '5'
54 = '6'
55 = '7'
56 = '8'
57 = '9'
```

//-----

4.1.3 N: Izpis abecede velikih črk

Izpišimo abecedo velikih črk. Izpis napišimo v zanki.

Namen: Izpis velikih črk abecede v zanki.

Izpis:

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

//-----

4.1.4 N: Izpis abecede malih črk

Izpišimo abecedo malih črk. Izpis napišimo v zanki.

Namen: Izpis malih črk abecede v zanki.

Izpis:

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

//-----

4.1.5 N: *getchar()*, *putchar()*

Program naj prebere vrstico, ki jo vpišemo do tipke »Enter« in naj jo izpiše. To naj poteka v zanki. Če je v prebrani vrstici znak '.', naj se delovanje programa konča.

Namen: Branje črk vrstice v zanki z *getchar()*, pisanje črk vrstice v zanki s *putchar()*.

Izpis:

```
Upisite besedilo (koncajte vpis z Enter, program pa s . (piko)):  
zelena mamba  
zelena mamba  
majhna strupena kaca.  
majhna strupena kaca.
```

//-----

4.1.6 N: 12 faktoriel

Napišimo program, ki izračuna in izpiše tabelo prvi 12 faktoriel (od 1 ! do 12 !).

npr. $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Namen: Računanje in izpis v zanki. Prejšnji rezultat uporabimo v izračunu trenutnega rezultata.

Izpis:

```
Tabela prvih 12 faktoriel:
  i          i!
  1          1
  2          2
  3          6
  4         24
  5        120
  6       720
  7      5040
  8     40320
  9    362880
 10   3628800
 11  39916800
 12 479001600
```

//-----

4.1.7 N: Tabela sinus[0..2π]

Tabelirajmo funkcijo $\sin(x)$ med 0° in 360° (2π radianov) s korakom 10° .

Namen: Uporaba funkcij iz matematične knjižnice. Ugotovitvi, da so argumenti teh funkcij tipa *double* (64 bit) in da je argument x funkcije $\sin(x)$ v radianih. Pretvorba med radiani in stopinjami, oblikovanje izpisa.

Izpis:

```
sin( 0) = 0.000000    sin(260) = -0.984808
sin(10) = 0.173648    sin(270) = -1.000000
sin(20) = 0.342020    sin(280) = -0.984808
sin(30) = 0.500000    sin(290) = -0.939693
sin(40) = 0.642788    sin(300) = -0.866025
sin(50) = 0.766044    sin(310) = -0.766044
sin(60) = 0.866025    sin(320) = -0.642788
sin(70) = 0.939693    sin(330) = -0.500000
sin(80) = 0.984808    sin(340) = -0.342020
sin(90) = 1.000000    sin(350) = -0.173648
sin(100) = 0.984808   sin(360) = -0.000000
```

//-----

4.1.8 N: Integral $\sin(x)$

Napišimo program za računanje integrala funkcije $\sin(x)$ po trapezni metodi v mejah $[0 .. 2\pi]$. Program naj sešteva predznačene ploščine oziroma produkte vrednosti $(\sin(x) + \sin(x + \Delta x)) / 2$ z razmaki Δx .

Namen: Računanje integrala funkcije po trapezni metodi oziroma predznačene ploščine pod krivuljo funkcije. Opaziti, da je računanje ploščine pod krivuljo funkcije, ki je simetrična glede na absciso, praktično neodvisno od velikosti Δx , če predznačeno ploščino računamo za celo periodo funkcije in če je perioda večkratnik Δx . Razlog je simetričnost funkcije – enako napako predznačene ploščine dobimo nad in pod absciso. Napaki se odštejeta. Če perioda ni večkratnik Δx , je napaka opazna.

Izpis:

```
p1 = -0.000000
```

//-----

4.1.9 N: Integral polinoma

Po trapezni metodi izračunajmo približek integrala funkcije $f(x) = x^2 + 5x + 3$ na intervalu $[0 .. 10]$ s korakom 0,001 (10000 korakov).

Namen: Računanje integrala funkcije oziroma predznačene ploščine pod krivuljo funkcije. Opaziti, da bo napaka rezultata odvisna od izbire velikosti Δx .

Izpis:

```
p1 = 613.333335
```

//-----

4.1.10 N: Fibonaccijevo zaporedje

Sestavimo program, ki izpiše prvih n členov Fibonaccijevega zaporedja (to je zaporedje, podano s predpisom: $f(1) = 1, f(2) = 1, f(n) = f(n-1) + f(n-2)$).

Namen: Računanje in izpis v zanki. Seznanitev s Fibonaccijevim zaporedjem.

Izpis:

```
Upisite stevilo členov: 12  
Členi zaporedja so: 1 1 2 3 5 8 13 21 34 55 89 144
```

//-----

4.1.11 N: Piramida

Sestavimo program, ki bo izpisal piramido velikosti n , kot je to prikazano v izpisu. Uporabnik vnese število vrstic n oziroma višino. Spodnja vrstica piramide naj začne skrajno levo.

Namen: Oblikovanje izpisa v zanki. Ugotoviti zakonitost pisanja splošne vrstice, ki jo potem izpisujemo v zanki s tekočimi indeksi.

Izpis:

```
Upisite visino piramide: 8
 *
 ***
 *****
 ****
 *****
 *****
 *****
 *****
 *****
 *****
 *****
```

//-----

4.1.12 N: Poštevanka 10 * 10

Vprogramirajmo izpis poštevanka od 1 do 10, z oznakami 1-10 zgoraj in ob strani.

Namen: Izračun in oblikovanje izpisa v zankah. Ena zanka naj bo za prvo vrsto, ena za drugo vrsto, ena za preostanek izpisa.

Izpis:

```
 1 1 2 3 4 5 6 7 8 9 10
 2 2 4 6 8 10 12 14 16 18 20
 3 3 6 9 12 15 18 21 24 27 30
 4 4 8 12 16 20 24 28 32 36 40
 5 5 10 15 20 25 30 35 40 45 50
 6 6 12 18 24 30 36 42 48 54 60
 7 7 14 21 28 35 42 49 56 63 70
 8 8 16 24 32 40 48 56 64 72 80
 9 9 18 27 36 45 54 63 72 81 90
10 10 20 30 40 50 60 70 80 90 100
```

//-----

4.1.13 N: Vsota števk

Napišimo program, ki izračuna vsoto števk vnesene številke. Vse številke naj bodo različne od 0. Vpis številke, izračun in izpis naj bodo v zanki, katere izvajanje prekinemo z vnosom številke 999.

Namen: Uporaba celoštevilskega deljenja in ostanka v zanki za izvedbo naloge.

Izpis:

```
Racunanje vsote števk številke.  
Upisite številko, 999 za konec: 12034  
Vsota števk številke 12034 je 10  
  
Upisite številko, 999 za konec: 999  
Vsota števk številke 999 je 27  
  
Konec.
```

//-----

4.1.14 N: e^x kot vrsta do (člen $< \epsilon$)

Sestavimo program, ki prebere realno število x in izračuna e^x s pomočjo spodnje potenčne vrste. Zadnji upoštevani člen potenčne vrste naj bo večji ali enak vnaprej predpisani konstanti ϵ (dovoljena napaka).

$$e^x = 1 + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^{n-1}}{(n-1)!}$$

Namen: Računanje v zanki. Iteracijski postopek zaključimo ob dovolj majhnem členu.

Izpis:

```
Racunamo e**x kot vrsto.  
Unesi x: 1  
e**1 z 14 cleni vrste je izracunan kot: 2.71828  
14 clen vrste je se >= 1.0E-10 (doloceni epsilon)
```

//-----

4.2 Rešitve (R)

4.2.1 R: Izpis ASCII znakov, [0 .. 255]

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int i;  
    puts ("");  
    for (i = 0; i <= 255; ++i)  
        printf("%d %c\n", i, i);  
  
    return 0;  
}  
  
//-----
```

4.2.2 R: Izpis ASCII znakov iz intervala

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int m, n, i;  
  
    printf("\nVpisite spodnjo mejo v ASCII tabeli: ");  
    scanf("%d", &m);  
    printf("Vpisite zgornjo mejo v ASCII tabeli: ");  
    scanf("%d", &n);  
  
    for (i=m; i<=n; i++)  
        printf("%d = '%c'\n", i, i);  
  
    printf("\n");  
  
    return 0;  
  
}  
  
//-----
```

4.2.3 R: Izpis abecede velikih črk

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int i;  
    puts("");  
    for (i = 65; i <= 90; ++i) {  
        //printf("%d %c ", i, i);  
        printf("%c ", i);  
    }  
  
    printf("\n");  
  
    return 0;  
}  
  
//-----
```

4.2.4 R: Izpis abecede malih črk

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int i;  
    puts("");  
    for (i = 97; i <= 122; ++i) {  
        //printf("%d %c ", i, i);  
        printf("%c ", i);  
    }  
    printf("\n");  
  
    return 0;  
}  
  
//-----
```

4.2.5 R: *getchar()*, *putchar()*

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    char c;  
  
    puts("\nVpisite besedilo (koncajte vpis z Enter, program pa s . (piko));");  
  
    do {  
        c=getchar();  
        putchar(c);  
    } while (c != '.');  
  
    puts("");  
  
    return 0;  
}  
  
//-----
```

4.2.6 R: 12 faktoriel

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int i, j = 1;  
    puts ("\nTabela prvih 12 faktoriel:\n");  
    puts (" i      i!\n");  
  
    for (i=1; i<= 12; i++) {  
        j = j * i;  
        printf (" %2i  %10i\n", i, j);  
    }  
  
    return 0;  
}  
  
//-----
```

4.2.7 R: Tabela $\sin[0 .. 2\pi]$

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    // pi = 180 stopinj, cos(pi) = -1  
    // kotne funkcije imajo argumente v radianih  
    // sin(x) argumenti: double, zato double Pi in izpis %9lf  
  
    int i;  
    double Pi = acos(-1);  
    puts("");  
    for (i = 0; i <= 360; i += 10) {  
        printf("sin(%3d) = %9lf\n", i, sin(Pi*i/180.0));  
    }  
  
    return 0;  
}  
  
//-----
```


4.2.8R: Integral $\sin(x)$

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    double pl=0, y;  
    int i;  
    double Pi = acos(-1);  
    int korak = 1; // stopinj  
  
    puts("");  
    for (i = 0; i <= 360-korak; i += korak)  
        pl += (sin(Pi*i/180.0) + sin(Pi*(i+korak)/180.0)) / 2 * korak;  
  
    printf("pl = %lf\n", pl);  
  
    return 0;  
}  
  
//-----
```

4.2.9 R: Integral polinoma

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    double korak = 0.001;  
    double pl=0, y1, y2, x;  
  
    for (x=0.0; x<=(10.0-korak); x+=korak) {  
        y1 = pow(x,2) + 5*x + 3;  
        y2 = pow(x+korak,2) + 5*(x+korak) + 3;  
        pl += (y1+y2)/2 * korak;  
    }  
    printf("\npl = %f\n", pl);  
  
    return 0;  
}  
  
//-----
```

4.2.10 R: Fibonaccijevo zaporedje

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int n, a, b, c;  
  
    printf("\nVpisite stevilo clenov: ");  
    scanf("%d", &n);  
  
    printf("Cleni zaporedja so:");  
    a = 1;  
    b = 0;  
    while (n-- > 0) { // n se postopno zmanjsuje  
                    // je pac navzdol tekoci stevec  
                    // razlikovati med --n in n--  
        c = a + b;  
        printf(" %d", c);  
        a = b;  
        b = c;  
        //n--;  
    }  
    printf("\n");  
  
    return 0;  
}  
//-----
```

4.2.11 R: Piramida

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int visina;  
    int i, j;  
  
    printf("\nVpisite visino piramide: ");  
    scanf("%d", &visina);  
  
    // najprej je potrebno ugotoviti pravilo:  
    // V i-ti vrstici je (visina - i) presledkov in (2 * i - 1) zvezdic.  
  
    for (i=1; i<=visina; i++) {  
  
        for (j=1; j<=visina-i; j++)  
            printf(" ");  
  
        for (j=1; j<=2*i-1; j++)  
            printf("*");  
  
        printf("\n");  
    }  
    return 0;  
}  
//-----
```

4.2.12 R: Poštevanka 10 * 10

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int i, j, k, l;  
  
    // prva vrsta  
    printf("\n |");  
    for(i=1; i<=10; i++)  
        printf(" %3d",i);  
    printf("\n");  
  
    // druga vrsta  
    for(j=1; j<=i; j++)  
        printf("----");  
    printf("\n");  
  
    // preostanek izpisa  
    for(k=1; k<=10; k++){  
        printf(" %2i| ", k);  
  
        for(l=1; l<=10; l++)  
            printf("%3i ",k*l);  
  
        printf("\n");  
    }  
    return 0;  
}  
//-----
```

4.2.13 R: Vsota števk

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int iVsotaStevk, n, iSprotni;  
    printf ("\nRacunanje vsote stevk stevilke.\n\n");  
  
    do {  
        printf ("Vpisite stevilko, 999 za konec: ");  
        scanf ("%i", &n);  
        //if (n==999) continue;  
  
        iVsotaStevk = 0; iSprotni = abs(n);  
        while (iSprotni) {  
            iVsotaStevk += iSprotni % 10;  
            iSprotni /= 10;  
        }  
        printf ("Vsota stevk stevilke %i je %i\n\n", n, iVsotaStevk);  
  
    } while (n != 999);  
  
    printf ("Konec.\n");  
    return 0;  
}  
//-----
```

4.2.14 R: e^x kot vrsta do (člen $< \epsilon$)

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
#pragma argsused  
int main(int argc, char* argv[]) {  
    #define EPS 1.0e-10  
    double x, vsota, clen;  
    int faktor, n;  
  
    puts("\nRacunamo e**x kot vrsto.");  
    printf("Vnesi x: ");  
    scanf("%lf", &x);  
  
    vsota = 0.0;  
    clen = 1.0;  
    n = 0;  
  
    faktor = 1;  
    while (fabs(clen) >= EPS) {  
        vsota += clen;  
        clen *= x;  
        clen /= faktor;  
        faktor++;  
        n++;  
    }  
  
    printf("e**%lg s %i clen vrste je izracunan kot: %g\n", x, n, vsota);  
    printf("%i clen vrste je se >= 1.0E-10 (doloceni epsilon)\n", n);  
    return 0;  
}  
//-----
```

5 Odločanje

Poudarki:

- Odločanje *if* (logični pogoj) {stavki};
- Odločanje *if* (logični pogoj) {stavki}; *else* {stavki};.
- Odločanje *switch* (spremenljivka) *case* (vrednost) {stavki};
- Zanka *while* (logični pogoj) {stavki}; in izhod iz zanke z *break*;
- Uporaba pogojnega operatorja *?* za odločanje.
- Tip *bool*.
- Dinamična dodelitev tipa spremenljivke za odločanje.
- Sestavljen logični pogoj.
- Logični pogoji z aritmetičnimi izrazi.
- Knjižnica *math*: *sqrt(x)*, *pow(x,z)*.

5.1 Naloge

5.1.1 N: Je zadnja številka enaka »0«?

Program naj ugotovi in izpiše, ali je zadnja številka vnešenega števila enaka »0«.

Namen: Uporaba *if()* ; *else* ; za odločanje.

Izpis:

```
Je zadnja številka v številu enaka 0?  
Upisite naravno število: 10  
Zadnja številka števila 10 je nič.
```

//-----

5.1.2 N: Celoštevilaska deljivost?

Program naj ugotovi in izpiše, ali je prvo vneseno število celoštevilsko deljivo z drugim vnesenim številom.

Namen: Stavki za odločanje in logični pogoji za odločanje.

Izpis:

```
Ali je prvo število celoštevilsko deljivo z drugim?  
Upisite dve celi števili, 999 999 za konec:  
20 10  
  
20 je celoštevilsko deljivo z 10  
Upisite dve celi števili, 999 999 za konec:  
999 999  
Konec.
```

//-----

5.1.3 N: Deljenje z realnim rezultatom

Delimo dve celi števili in kvocient izračunajmo na tri decimalna mesta.

Namen: Odločitveni stavki, izračun je del stavka izpisa, dinamična dodelitev tipa spremenljivke v računskem izrazu za preklop s celoštevilskega deljenja v deljenje realnih števil.

Izpis:

```
Deljenje dveh celih števil, kvocient je izračunan na tri decimalna mesta.  
  
Upisite dve celi števili, 999 999 za konec:  
12 5  
12 deljeno z 5 je 2.400  
  
Upisite dve celi števili, 999 999 za konec:  
999 999  
Konec.
```

//-----

5.1.4 N: Značilke vnosov števil

Naredimo program, ki bere števila, dokler ne vnesemo števila 0. Program naj sprti izpisuje največje število, najmanjše število ter povprečje števil.

Namen: Uporaba stavkov *while()* in *break* za implementacijo neskončne zanke s pogojenim izhodom, uporaba pogojnega operatorja *?* za odločanje, dvojna dodelitev vrednosti v enem stavku, dinamična dodelitev tipa spremenljivke.

Dinamična dodelitev tipa spremenljivke: Tip spremenljivke določimo z deklaracijo, npr:

```
int spremenljivka1 = 50, spremenljivka2 = 20;
```

Tekom izvajanja programa lahko spremenljivki sprti, torej dinamično, začasno spremenimo tip z zapisom *(float)spremenljivka1*.

```
float rezultat1 = spremenljivka1 / spremenljivka2;  
rezultat1 je 2 - celoštevilsko deljenje
```

```
float rezultat2 = (float)spremenljivka1 / spremenljivka2;  
rezultat2 je 2,5 - deljenje realnih števil.
```

Izpis:

```
Unesi celo stevilo, 0 za konec: 3  
min = 3, max = 3, povp = 3  
Unesi celo stevilo, 0 za konec: 5  
min = 3, max = 5, povp = 4  
Unesi celo stevilo, 0 za konec: 122  
min = 3, max = 122, povp = 43.3333  
Unesi celo stevilo, 0 za konec: 0  
Konec.
```

//-----

5.1.5 N: Je leto prestopno?

Napišimo program za ugotavljanje prestopnosti vnešenega leta. Leto je prestopno, če je celoštevilsko deljivo s 4 in hkrati ni celoštevilsko deljivo s 100 ali pa je celoštevilsko deljivo s 400.

Namen: Pisanje sestavljenega logičnega pogoja.

Izpis:

```
Unesi letnico: 2016
Leto je prestopno.
```

```
Unesi letnico: 2014
Leto ni prestopno.
```

//-----

5.1.6 N: Trikotnik a,b,c obstoja?

Za trikotnik vpišemo dolžine stranic a , b in c . Program naj preveri, če trikotnik obstoja. Če obstoja, naj program izračuna obseg in ploščino.

Namen: Iz npr. <http://sl.wikipedia.org/wiki/Trikotnik> poiskati potrebne napotke in jih zapisati v odločitvene stavke in v računski izraz.

Izpis:

```
Trikotnik, vpišite dolzino stranice a: 3
Trikotnik, vpišite dolzino stranice b: 4
Trikotnik, vpišite dolzino stranice c: 5

Tak trikotnik obstoja.
Obseg = 12 enot.
Ploscina = 6 kvadratnih enot.
```

```
Trikotnik, vpišite dolzino stranice a: 1
Trikotnik, vpišite dolzino stranice b: 2
Trikotnik, vpišite dolzino stranice c: 0.5

Tak trikotnik ne obstoja.
```

//-----

5.1.7 N: Kalkulator + - * /

Napišimo kalkulator za seštevanje, odštevanje, množenje, deljenje celih števil z rezultatom v obliki realnega števila. V primeru deljenja z 0 naj se izpiše »Napaka, deljenje z 0«.

Namen: Uporaba *switch case* odločitvenega stavka. Dinamična sprememba tipa spremenljivke za izračun rezultata v obliki realnega števila.

Izpis:

```
Unesite prvo celostevilsko stevilo: 12
Unesite drugo celostevilsko stevilo: 20
Unesite racunsko operacijo ( +, -, *, / ): /
12 / 20 = 0.600000
```

//-----

5.1.8 N: Sklanjanje za vnos

Program naj vpraša po številu vijakov. Vpišete nenegativno število. Program pravilno oblikuje izpis kot:

Mehanik ne privija vijakov.

Mehanik privija 1 vijak.

Mehanik privija 102 vijaka.

Namen: Uporaba *switch case* odločitvenega stavka. Najprej razmislite o potrebah in možnostih stavčnega oblikovanja, da bo posledično malo kode, kar je sinonim za učinkovito kodo.

Izpis:

```
Unesite celo število vijakov: 0  
Mehanik ne privija vijakov.
```

```
Unesite celo število vijakov: 1  
Mehanik privija 1 vijak.
```

```
Unesite celo število vijakov: 2  
Mehanik privija 2 vijaka.
```

```
Unesite celo število vijakov: 3  
Mehanik privija 3 vijake.
```

```
Unesite celo število vijakov: 4  
Mehanik privija 4 vijake.
```

```
Unesite celo število vijakov: 5  
Mehanik privija 5 vijakov.
```

```
Unesite celo število vijakov: 102  
Mehanik privija 102 vijaka.
```

//-----

5.1.9 N: Je n praštevilo?

Napišimo program za preverjanje, ali je dano naravno število (pozitivno celo število) praštevilo.

Praštevilo je deljivo le z 1 in samo s sabo.

Npr. prvih 20 pozitivnih praštevil: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

Namen: Izbrati ustrezen algoritem in ga v zanki implementirati.

Izpis:

```
Unesi naravno stevilo: 7
Stevilo 7 je prastevilo.
```

//-----

5.1.10 N: Produkt ne-nič števk

Sestavimo program, ki prebere pozitivno celo številko in izračuna produkt neničelnih števk.

Namen: Sestavljanje algoritma s celoštevilskm deljenjem in z operatorjem % za računanje ostanka pri celoštevilskem deljenju. Računanje v zanki.

Izpis:

```
Upisite pozitivno celo številko: 12034
Produkt nenicelnih števk je 24.
```

//-----

5.1.11 N: Izpis števk z besedami

Napišimo program, kjer najprej vpišemo pozitivno celo število, nato pa naj se števke izpišejo z besedami od zadaj naprej. Za končanje programa vpišemo 999.

Namen: Uporaba *switch case*. Program spremenite, da bo izpis števk z besedami v vrstnem redu števk.

Izpis:

```
Upisite pozitivno celo število z največ 10 znaki. 999 za konec:  
567801  
ena nič osem sedem šest pet  
999  
Konec.
```

//-----

5.2 Rešitve (R)

5.2.1 R: Je zadnja številka enaka »0«?

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
  
int main(int argc, char* argv[])  
{  
    int n;  
    puts("\nJe zadnja številka v številu enaka 0?");  
    printf("Vpisite naravno število: ");  
    scanf("%d", &n);  
    if (n % 10 == 0)  
        printf("Zadnja številka števila %d je nič.\n", n);  
    else  
        printf("Zadnja številka števila %d ni nič.\n", n);  
  
    return 0;  
}  
  
//-----
```


5.2.2 R: Celoštevilska deljivost?

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int iVecji, iManjsi;  
  
    printf ("\nAli je prvo stevilo celostevilsko deljivo z drugim?\n");  
  
    while (true) {  
        printf ("Vpisite dve celi stevili, 999 999 za konec:\n");  
        scanf ("%i %i", &iVecji, &iManjsi);  
        if (iVecji == 999 && iManjsi == 999) break;  
  
        // dinamicno dodeljevanje tipa spremenljivke:  
        // kot dobra praksa, tu deluje tudi brez (bool)  
        if (!(bool)(iVecji % iManjsi))  
            printf ("\n%i je celostevilsko deljivo z %i\n", iVecji, iManjsi);  
        else  
            printf ("\n%i ni celostevilsko deljivo z %i\n", iVecji, iManjsi);  
    }  
  
    printf ("Konec.\n");  
    return 0;  
}  
//-----
```

5.2.3 R: Deljenje z realnim rezultatom

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[]) {  
    int Divident = 0, Divisor = 0;  
    printf ("\nDeljenje dveh celih stevil, kvocient je izracunan na tri decimalna  
mesta.\n\n");  
  
    while (true) {  
        printf ("\nVpisite dve celi stevili, 999 999 za konec:\n");  
        scanf ("%i %i", &Divident, &Divisor);  
        if (Divident == 999 && Divisor == 999) break;  
        if (!Divisor) {  
            printf ("\nDeljenje z niclo, ni izracunljivo!");  
            continue;  
        }  
        else  
            // dinamicna dodelitev tipa spremenljivke (float):  
            // brez tega rezultat ni realno stevilo, ampak je celo stevilo  
            printf ("%i deljeno z %i je %.3f\n", Divident, Divisor, (float)Divident /  
Divisor);  
    }  
  
    printf ("\nKonec.\n");  
    return 0;  
}  
//-----
```

5.2.4 R: Značilke vnosov števil

```
//-----  
#include <stdio.h>  
#include <math.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int n, x, min, max, vsota;  
  
    n = 0;  
    vsota = 0;  
    puts("");  
    while (true) { // neskončna zanka, true je 1, definirano v stdbool.h  
        printf("Vnesi celo število, 0 za konec: ");  
        scanf("%d", &x);  
  
        if (x == 0) break; // zapusti zanko in zaključi program  
  
        n++;  
        vsota += x;  
  
        if (n == 1) {  
            min = max = x; // dvojna dodelitev vrednosti, redko videno  
        } else {  
            //if (x < min) min = x;  
            //if (x > max) max = x;  
            min = (x < min) ? x : min;  
            max = (x > max) ? x : max;  
        }  
        // dinamična dodelitev tipa float v izpisu za deljenje realnih  
        // namesto celih števil
```

```
    printf("min = %i, max = %i, povp = %g\n", min, max, (float)vsota/n);  
}  
puts("Konec.");  
return 0;  
}  
//-----
```

5.2.5 R: Je leto prestopno?

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int leto;  
  
    printf("Vnesi letnico: ");  
    scanf("%d", &leto);  
  
    if ((leto%4 == 0 && leto%100 != 0) || leto%400 == 0) {  
        printf("Leto je prestopno.\n");  
    } else {  
        printf("Leto ni prestopno.\n");  
    }  
    return 0;  
}  
//-----
```

5.2.6 R: Trikotnik a, b, c obstoja?

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
#pragma argsused  
  
int main(int argc, char* argv[])  
{  
    // veljati mora trikotniska neenakost: pozitivne stranice in  
    // vsota katerihkoli dveh stranic < tretja stranica  
  
    // http://sl.wikipedia.org/wiki/Trikotnik  
    // enostaven izracun ploscine, Heronova formula:  
    //  $a = \sqrt{\text{pol\_obseg} * (\text{pol\_obseg} - a) * (\text{pol\_obseg} - b) * (\text{pol\_obseg} - c)}$   
  
    double a, b, c;  
  
    printf("\nTrikotnik, vpisite dolzino stranice a: ");  
    scanf("%lf", &a);  
  
    printf("Trikotnik, vpisite dolzino stranice b: ");  
    scanf("%lf", &b);  
  
    printf("Trikotnik, vpisite dolzino stranice c: ");  
    scanf("%lf", &c);  
  
    if (a>0 && b>0 && c>0 && a+b>c && a+c>b && b+c>a) {  
        double o, pl, s;  
  
        o = a + b + c;  
        s = 0.5 * o;  
        pl = sqrt(s*(s-a)*(s-b)*(s-c));  
  
        printf("\nTak trikotnik obstoja.\n");  
    }
```

```
printf("Obseg = %g enot.\n", o);  
printf("Ploscina = %g kvadratnih enot.\n", pl); }  
  
else  
    printf("\nTak trikotnik ne obstoja.\n");  
return 0;  
}  
  
//-----
```

5.2.7 R: Kalkulator + - * /

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int x;  
    int y;  
    float z = 0;  
    int err = 0;  
    char c;  
    printf("\nVnesite prvo celostevilsko stevilo: ");  
    scanf("%d", &x);  
    printf("Vnesite drugo celostevilsko stevilo: ");  
    scanf("%d", &y);  
    printf("Vnesite racunsko operacijo ( +, -, *, / ): ");  
    scanf("%*c%c", &c);  
    // scanf("%c", &c);  
  
    switch (c) {  
        case '+': z = x + y; break;  
        case '-': z = x - y; break;  
        case '*': z = x * y; break;  
        case '/': if (y==0) err=1; else z = (float)x / y; break;  
  
        default: err = 2; break;  
    }  
  
    if (err == 0)  
        printf("%i %c %i = %f", x,c,y,z);  
    else
```



```
switch (err) {
    case 1 : printf("Napaka: deljenje z 0."); break;
    case 2 : printf("Napaka: neznana racunska operacija."); break;

    default: printf("Napaka: (%d)", err); break;
}

return 0;
}
//-----
```

5.2.8 R: Sklanjanje za vnos

```
//-----  
#include <stdio.h>  
#include <math.h>  
#include <stdbool.h>  
#pragma hdrstop  
  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int x;  
    printf("\nVnesite celo stevilo vijakov: ");  
    scanf("%d", &x);  
  
    if (x>0)  
        switch (x % 100) {  
            case 1: printf("Mehanik privija %d vijak.", x); break;  
            case 2: printf("Mehanik privija %d vijaka.", x); break;  
            case 3:  
            case 4: printf("Mehanik privija %d vijake.", x); break;  
  
            default: printf("Mehanik privija %d vijakov.", x); break;  
        }  
    else  
        printf("Mehanik ne privija vijakov.");  
  
    return 0;  
}  
  
//-----
```

5.2.9 R: Je n praštevilo?

```
//-----  
#include <stdio.h>  
#include <math.h>  
#include <stdbool.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int n, i, stDel;  
  
    printf("\nVnesi naravno stevilo: ");  
    scanf("%d", &n);  
  
    // presteli bomo delitelje stevila n, enostavno in neoptimirano  
  
    stDel = 0;  
    i = 1;  
    while (i <= n) {  
        if (n % i == 0) stDel = stDel + 1;  
        i++;  
    }  
  
    if (stDel == 2) {  
        printf("Stevilo %d je prastevilo.\n", n);  
    } else {  
        printf("Stevilo %d ni prastevilo.\n", n);  
    }  
  
    return 0;  
}  
  
//-----
```

5.2.10 R: Produkt ne-nič števk

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int n, stevka;  
    int produkt = 1;  
  
    printf("\nVpisite pozitivno celo stevilko: ");  
    scanf("%d", &n);  
  
    while (n > 0) {  
        stevka = n % 10;  
        if (stevka != 0)  
            produkt *= stevka;  
        n = n / 10;  
    }  
  
    printf("Produkt nenicelnih stevk je %d.\n", produkt);  
  
    return 0;  
}  
  
//-----
```

5.2.11 R: Izpis števk z besedami

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int iVneseni, iStevilo, iOstaneK;  
  
    printf ("\n\nVpisite pozitivno celo stevilko z največ 10 znaki. 999 za konec:\n");  
  
    while (true) {  
        scanf("%i", &iVneseni);  
        if (iVneseni == 999) {  
            printf ("\nKonec.\n");  
            return 0; }  
  
        iStevilo = abs(iVneseni);  
        while (iStevilo) {  
            iOstaneK = iStevilo % 10;  
            switch (iOstaneK) {  
                case 0: printf ("nic "); break;  
                case 1: printf ("ena "); break;  
                case 2: printf ("dve "); break;  
                case 3: printf ("tri "); break;  
                case 4: printf ("stiri "); break;  
                case 5: printf ("pet "); break;  
                case 6: printf ("sest "); break;  
                case 7: printf ("sedem "); break;  
                case 8: printf ("osem "); break;  
                case 9: printf ("devet "); break;  
                default: break;  
            }  
        }  
    }  
}
```

```
    }
    iStevilo /= 10;
  }
  puts("");
}

printf ("\nKonec.\n");

return 0;
}
//-----
```

6 Polja

Poudarki:

- Deklaracija polja.
- Inicializacija vrednosti polja z vpisom vrednosti posameznih elementov, v zanki, z dodelitvijo vrednosti spominskim lokacijam polja – *memset()*.
- Polja s stalnimi vrednostmi – *const*.
- Podvajanje vrednosti polja z dodeljevanjem vrednosti in s kopiranjem vrednosti spominskih lokacij polja – *memcpy()*.
- Deklaracija in uporaba dvodimenzijskega polja spremenljivk oziroma matrike spremenljivk.
- Vpisi in izpisi vrednosti elementov polj.
- Vpisi in izpisi vrednosti elementov polj v zanki.
- Algebra z elementi polj.

6.1 Naloge

6.1.1 N: Vnosi in povprečje

Napišimo program, kjer vpišemo do 10 realnih števil. Vpis naj se konča po 10 vpisanih številih ali po vpisu "999".

Izpiše naj se povprečna vrednost vnešenih števil.

Za hrambo vnesenih števil uporabimo polje realnih števil.

Namen: Shranjevanje števil v polju. Vpis in uporaba polja v zanki. Inicializacija polja.

Izpis:

```
Upis 10 realnih števil, vpisite 1 stevilo, 999 za konec:
32.567
Upis 10 realnih števil, vpisite 2 stevilo, 999 za konec:
65748.98765
Upis 10 realnih števil, vpisite 3 stevilo, 999 za konec:
999
Upisanih števil 2, poprecna vrednost je 32890.7773
Konec.
```

//-----

6.1.2 N: Vnosi in dva izpisa

Napišimo program, kjer uporabnik najprej vpiše podatke v polje celih števil. Nato se števila izpišejo v vrstnem redu vnosa in v obratnem vrstnem redu.

Namen: Vpis v polje števil. Branje in izpis polja v zanki v obe smeri.

Izpis:

```
Unos celih števil, izpis od prvega do zadnjega in obratno.  
Unesite stevilo celih števil (med 0 in 10): 6  
1. stevilo: 2  
2. stevilo: 4  
3. stevilo: 6  
4. stevilo: 8  
5. stevilo: 10  
6. stevilo: 12  
Izpis od prvega do zadnjega: 2 4 6 8 10 12  
Izpis od zadnjega do prvega: 12 10 8 6 4 2
```

//-----

6.1.3 N: Kopiranje polja v polje

Naredimo dve polji, vsako dolžine 6 spremenljivk celoštevilskega tipa.

Prvo polje inicializirajmo s poljubno izbranimi vrednostmi.

Izpišimo vrednosti obeh polj.

Kopirajmo v zanki prvo polje v drugo polje.

Izpišimo vrednosti obeh polj.

Namen: Inicializacija vrednosti polja po posameznih elementih ali s

postavitvijo spominske lokacije, *memset()*. Kopiranje vrednosti polja po

posameznih elementih ali s kopiranjem spominske lokacije, *memcpy()*.

Izpis vrednosti polja v zanki.

Izpis:

```
Pred kopiranjem:
A: 9 1 5 6 2 8
B: 844545285 0 0 4202502 4199233 844555749

B po postavitvi zacetnih vrednosti:
0 0 0 0 0 0

Po kopiranju:
A: 9 1 5 6 2 8
B: 9 1 5 6 2 8
```

//-----

6.1.4 N: Iskanje v polju

Napišimo program, kjer uporabnik najprej napiše, koliko celih števil bo vpisal. Nato jih vpiše. Nato izbere vrednost števila, ki ga išče. Najdeno je prvič vpisano iskano število ali pa program pove, da iskanega števila ni v tabeli.

Namen: Iskanje v polju spremenljivk v zanki. Vpis polja v zanki.

Izpis:

```
Unesi stevilo števil (med 0 in 100): 6
0. stevilo: 5
1. stevilo: 6
2. stevilo: 5
3. stevilo: 6
4. stevilo: 56
5. stevilo: 56
Unesi iskano stevilo, 0 za konec: 56
Število 56 se prvič pojavi na 5. mestu (mesto z indeksom 4).
Unesi iskano stevilo, 0 za konec: 0
```

//-----

6.1.5 N: Sprememba številске baze

Napišimo program za pretvorbo števila iz desetiške baze v poljubno bazo med vključno dvojiško in šestnajstiško.

Namen: Uporaba elementov polja za oblikovanje rezultata in izpisa. Razmislek o številskih bazah in o univerzalnem algoritmu za pretvorbo iz desetiške v poljubno številsko bazo.

Izpis:

```
Pretvorba števila iz desetiške baze v bazo n.
Unesite število, 999 za izhod? 111
Baza? 2
Pretvorjeno število = 1101111

Unesite število, 999 za izhod? 10
Baza? 2
Pretvorjeno število = 1010

Unesite število, 999 za izhod? 10
Baza? 16
Pretvorjeno število = A

Unesite število, 999 za izhod? 999
Konec.
```

//-----

6.1.6 N: Fibonaccijeva števila

Izračunajmo in izpišimo prvih 20 zaporednih Fibonaccijevih števil.

Rekurzivna definicija zaporedja je: $f(1) = 0, f(2) = 1, f(n) = f(n-1) + f(n-2)$

Namen: Uporaba polja spremenljivk kot spominskih lokacij med izračunom in nadaljnjim procesiranjem. Izračunamo vsa števila, shranimo jih v polje spremenljivk, in jih v nadaljevanju programa izpišemo.

Izpis:

```
Fibonaccijevo zaporedje: 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

//-----

6.1.7 N: Koliko bankovcev?

Program naj izračuna, koliko katerih bankovcev je potrebnih za plačilo vnešenega zneska.

Namen: Polje spremenljivk uporabimo kot spominske lokacije z vrednostmi bankovcev. Z deljenjem ostanka zneska s trenutno izbiro bankovca ugotovimo število potrebnih bankovcev trenutne izbire.

Izpis:

```
Uneseni znesek razdelimo po evrskih bankovcih.  
Upisite znesek, 999 za konec: 1235  
2 x 500 eur  
1 x 200 eur  
1 x 20 eur  
1 x 10 eur  
1 x 5 eur  
  
Upisite znesek, 999 za konec: 999  
Konec.
```

//-----

6.1.8 N: Transponiranje matrice

Transponiramo dvodimenzionalno matriko. (Zamenjamo vrstice in stolpce).

Namen: Uporaba dvodimenzijskega polja. Manipulacija vrednosti dvodimenzijskega polja v zankah.

izpis:

```
Unesite dimenzijo kvadratne matrice, 999 za konec: 3
a[1,1]=1
a[1,2]=2
a[1,3]=3
a[2,1]=4
a[2,2]=5
a[2,3]=6
a[3,1]=7
a[3,2]=8
a[3,3]=9

Matrika
  1  2  3
  4  5  6
  7  8  9

Transponirana matrika
  1  4  7
  2  5  8
  3  6  9

Unesite dimenzijo kvadratne matrice, 999 za konec: 999
Konec.
```

//-----

6.2 Rešitve (R)

6.2.1 R: Vnosi in povprečje

```
//-----  
#include <stdio.h>  
#include <mem.h>  
#include <stdbool.h>  
#include <string.h>  
#pragma hdrstop  
//-----  
#pragma argsused  
int main(int argc, char* argv[]) {  
    #define DIM 10  
    int i, j;  
    float fTekoce;  
  
    float fPovprecje = 0, fStevila[10];  
    memset (fStevila, 0, 10*4);    // float 10 lokacij * 4 * 8 bitov  
  
    for (i = 0; i < DIM; i++) {  
        printf ("\nVpis 10 realnih stevil, vpisite %i stevilo, 999 za konec:\n", i+1);  
        scanf("%f", &fTekoce);  
        if (fTekoce == 999) break;  
        fStevila[i] = fTekoce;  
    }  
  
    for (j = 0; j < i; j++)  
        fPovprecje += fStevila[j];  
  
    if (i>0)  
        fPovprecje /= i;  
    printf ("Vpisanih stevil %i, povprečna vrednost je %.4f\nKonec.\n", i,  
fPovprecje);  
    return 0;  
}  
//-----
```

6.2.2 R: Vnosi in dva izpisa

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define MAX 10  
    int polje[MAX+1]; // izogib rabi prve spremenljivke - tabela[0]  
    int stStevil, i;  
  
    puts("\nVnos celih števil, izpis od prvega do zadnjega in obratno.");  
    printf("Vnesite število celih števil (med 0 in %d): ", MAX);  
    scanf("%d", &stStevil);  
    for (i=1; i<=stStevil; i++) {  
        printf("%i. število: ", i);  
        scanf("%i", &polje[i]);  
    }  
  
    printf("Izpis od prvega do zadnjega:");  
    for (i=1; i<=stStevil; i++) printf(" %d", polje[i]);  
    printf("\nIzpis od zadnjega do prvega:");  
    for (i=stStevil; i>=1; i--) printf(" %d", polje[i]);  
    printf("\n");  
  
    return 0;  
}  
//-----
```


6.2.3 R: Kopiranje polja v polje

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define DIM 6  
    int A[DIM]={9,1,5,6,2,8};  
    int B[DIM];  
    int i;  
  
    printf("\nPred kopiranjem:\nA: ");  
  
    for (i = 0; i <= DIM-1; ++i)  
        printf("%d ",A[i]);  
  
    printf("\nB: ");  
    for (i = 0; i <= DIM-1; ++i)  
        printf("%d ",B[i]);  
    printf("\n");  
  
    printf("\nB po postavitvi zacetnih vrednosti: \n");  
  
    for (i = 0; i <= DIM-1; ++i)  
        B[i]=0;  
    // memset(B, 0, DIM*4); // alternativa  
  
    for (i = 0; i <= DIM-1; ++i)  
        printf("%d ",B[i]);  
  
    printf("\n\nPo kopiranju: ");
```

```
for (i = 0; i <= DIM-1; ++i)
    B[i]=A[i];
// memcpy(B, A, DIM*4); // alternativa
```

```
printf("\nA: ");
for (i = 0; i <= DIM-1; ++i)
    printf("%d ",i[A]);
```

```
printf("\nB: ");
for (i = 0; i <= DIM-1; ++i)
    printf("%d ",B[i]);
```

```
return 0;
}
```

```
//-----
```

6.2.4 R: Iskanje v polju

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define MAX 100  
    int polje[MAX];  
    int stStevil, n, i;  
  
    printf("\nVnesi stevilo stevil (med 0 in %d): ", MAX);  
    scanf("%d", &stStevil);  
    for (i=0; i<stStevil; i++) {  
        printf("%i. stevilo: ", i);  
        scanf("%i", &polje[i]);  
    }  
  
    while (true) {  
        printf("Vnesi iskano stevilo, ""0"" za konec: ");  
        scanf("%d", &n);  
        if (n == 0) break;  
  
        for (i=0; i<stStevil; i++)  
            if (polje[i] == n)  
                break;  
  
        if (i == stStevil)  
            printf("Stevila %d ni v tabeli.\n", n);  
        else  
            printf("Stevilo %d se prvič pojavi na %i. mestu (mesto z indeksom %i).\n", n,  
i+1, i);  
    }  
}
```

```
}// while
```

```
return 0;
```

```
}
```

```
//-----
```

6.2.5 R: Sprememba številske baze

```
//-----  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    const char stevkeBaze[16] = {  
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};  
    int spremenjenoStevilo[64];  
    long int vnesenoStevilo;  
    int naslednjaStevka, baza, index = 0;  
  
    puts("\n Pretvorba stevila iz desetiske baze v bazo n.");  
    while (true) {  
        index=0;  
        printf ("\n Vnesite stevilo, 999 za izhod? ");  
        scanf ("%ld", &vnesenoStevilo);  
        if (vnesenoStevilo == 999) break;  
  
        printf ("Baza? ");  
        scanf ("%i", &baza);  
  
        do {  
            spremenjenoStevilo[index] = vnesenoStevilo % baza;  
            ++index;  
            vnesenoStevilo = vnesenoStevilo / baza;  
        }  
        while ( vnesenoStevilo != 0 );  
  
        printf ("Pretvorjeno stevilo = ");
```

```
for (--index; index >= 0; --index ) {  
    naslednjaStevka = spremenjenoStevilo[index];  
    printf ("%c", stevkeBaze[naslednjaStevka]);  
}  
printf ("\n");  
}
```

```
puts("\nKonec.");
```

```
return 0;  
}
```

```
//-----
```

6.2.6 R: Fibonaccijeva števila

```
//-----  
#include <stdio.h>  
#include <math.h>  
#include <stdbool.h>  
#include <string.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define DIM 20  
    int fib[DIM];  
    int i;  
  
    fib[0] = fib[1] = 1;  
    for (i=2; i<DIM; i++) {  
        fib[i] = fib[i-1] + fib[i-2];  
    }  
  
    printf("\nFibonaccijevo zaporedje:");  
    for (i=0; i<DIM; i++) {  
        printf(" %d", fib[i]);  
    }  
    printf("\n");  
  
    return 0;  
}  
//-----
```

6.2.7 R: Koliko bankovcev ?

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[]) {  
    #define MAX 10  
    #define LEN 20  
    const int euroBankovec[] = {500, 200, 100, 50, 20, 10, 5, 2, 1};  
    int znesek, i;  
  
    puts("\nVneseni znesek razdelimo po evrskih bankovcih.");  
  
    while (true) {  
        printf("\nVpisite znesek, 999 za konec: ");  
        scanf("%i", &znesek);  
        if (znesek == 999) break; // izpad iz zanke  
  
        i = -1;  
        while (znesek > 0) {  
            i++;  
            if (euroBankovec[i] > znesek) continue; // preskok ostanka zanke v tej  
iteraciji  
            printf("%i x %i eur\n", znesek/euroBankovec[i], euroBankovec[i]);  
            znesek %= euroBankovec[i];  
        }  
    }  
    puts("\nKonec.");  
    return 0;  
}  
//-----
```


6.2.8 R: Transponiranje matrice

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define MAX 6  
    int a[MAX][MAX];  
    int n;  
    int i, j, m;  
  
    while (true) {  
        do {  
            printf("\nVnesite dimenzijo kvadratne matrice, 999 za konec: ");  
            scanf("%d", &n);  
            if (n == 999) {  
                puts("Konec."); return 0;  
            }  
        } while (n < 0 || n > MAX);  
  
        for (i = 0; i < n; i++)  
            for (j = 0; j < n; j++)  
            {  
                printf("a[%d,%d]=", i+1, j+1);  
                scanf("%d", &a[i][j]);  
            }  
  
        printf("\nMatrika \n");  
        for (i = 0; i < n; ++i) {  
            for (j = 0; j < n; ++j)  
                printf("%3d ", a[i][j]);  
            printf("\n");  
        }  
    }  
}
```

```

}
printf("\n");

for (i = 0; i < n; ++i)
    for (j = i; j < n; ++j) {
        m=a[i][j];
        a[i][j]=a[j][i];
        a[j][i]=m;
    }

printf("\nTransponirana matrika \n");
for (i = 0; i < n; ++i) {
    for (j = 0; j < n; ++j)
        printf("%3d ",a[i][j]);
    printf("\n");
}
}
return 0;
}
//-----

```

7 Funkcije

Poudarki:

- Definicija funkcije.
- Deklaracija funkcijskega prototipa.
- Strukturiranje programa v funkcije.
- Prenos vrednosti spremenljivk v funkcijo prek parametrov funkcije in prenos rezultata iz funkcije prek povratne vrednosti funkcije.
- Uporaba polja spremenljivk za parameter funkcije.
- Prenos vrednosti v/iz funkcije prek polja spremenljivk – parametra funkcije.
- Uporaba globalne spremenljivke v funkciji.
- Uporaba lokalne avtomatske spremenljivke v funkciji.
- Uporaba lokalne statične spremenljivke v funkciji.
- Rekurzivna funkcija.

7.1 Naloge

7.1.1 N: Vsota števil

Program naj vpraša po vnosu dveh števil. Števili naj sešteje in naj izpiše rezultat. Naj bo strukturiran v funkcije:

```
int preberi(void);  
int vsota (int a, int b);  
void izpiši (int a);
```

Namen: Strukturiranje programa v funkcije. Prenos spremenljivk prek parametrov funkcije in prenos rezultata prek povratne vrednosti funkcije.

Izpis:

```
Unesi celo stevilo: 23  
Unesi celo stevilo: 45  
Vsota je 68.
```

//-----

7.1.2 N: Parameter in spremenljivka

Napišimo funkcijo *void zamenjaj(int a, int b)*; Uporabimo jo v programu. Izpišimo spremenljivki *a* in *b* pred in po klicu funkcije.

Namen: Funkcija *void zamenjaj(int a, int b)* dejansko zamenja *a* in *b*. Sprememba se ne prenese nazaj v program prek parametrov *a* in *b* funkcije. Parametri funkcije predstavljajo enosmerno pot v funkcijo, rezultatov funkcije ne moremo vračati prek parametrov funkcije. Več o tem (navidezna izjema, dejansko tudi tam parametrov ne spreminjamo) v poglavju o kazalcih.

Izpis:

```
1, 2  
1, 2
```

//-----

7.1.3 N: Globalni spremenljivki

Napišimo funkcijo *void zamenjaj(void)*; Funkcija med samo zamenja vrednosti spremenljivk *a* in *b*. Uporabite jo v programu z globalnima spremenljivkama *a* in *b*. Izpišite spremenljivki *a* in *b* pred in po klicu funkcije.

Namen: Globalno spremenljivko beremo in pišemo globalno – v glavnem programu in v vseh funkcijah. Z globalnimi spremenljivkami se lahko izognemo prenašanju vrednosti spremenljivk v funkcije in rezultatov nazaj v glavni program ali v kličočo funkcijo. Slabost globalnih spremenljivk je, da jih lahko od povsod spreminjamo, s čimer so osiromašeni koncepti lokalnosti, modularne urejenosti programa in samozadostnosti funkcij.

Izpis:

```
1, 2  
2, 1
```

//-----

7.1.4 N: Rešitev kvadratne enačbe

Napišimo program za računanje rešitev kvadratne enačbe $ax^2 + bx + c = 0$.

Najprej izračunamo diskriminanto $D = b^2 - 4ac$. Če je $D < 0$, naj program izpiše, da sta rešitvi kompleksni, sicer pa naj izračuna obe rešitvi $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$.

Program strukturirajmo v funkcije `main()`, `kvadratniKoren()` in kar še potrebujemo brez uporabe matematične knjižnice (funkcije iz `math.h`).

Potem še napišimo isti program z uporabo funkcij iz matematične knjižnice.

Namen: Strukturiranje programa v funkcije. Prenos parametrov v funkcije, vrnitev rezultata funkcije. Razmislje o izdelavi algoritma, kadar je znana zahteva, postopkov pa je lahko več (v tem primeru računanje kvadratnega korena).

Izpis:

```
Racunamo korene kvadratne enacbe:
Unesite koeficiente a, b in c, 0 0 0 za konec:
2 8 4
Resitvi za 2.0000, 8.0000, 4.0000 sta: -0.585786 -3.414214
Unesite koeficiente a, b in c, 0 0 0 za konec:
0 0 0
Konec.
```

//-----

7.1.5 N: Funkcija Največji Skupni Delitelj (NSD)

Izračunajmo NSD v funkciji `int NSD(int a, int b)` in napišimo program, ki vpraša po obeh številih in izpiše NSD.

Funkcija NSD mora delovati tudi z nepričakovanimi številskimi vnosi (kot je npr. ničla).

Namen: Poiskati ali narediti algoritem, napisati funkcijo za izračun NSD in jo uporabiti v programu, ki omogoči vnos celih števil in izpis v zanki.

Izpis:

```
Majvecji skupni delitelj (NSD) dveh celih števil.  
Unesi 1. stevilo, -1 za konec: 320  
Unesi 2. stevilo, -1 za konec: 100  
NSD(320, 100) = 20  
  
Unesi 1. stevilo, -1 za konec: -1  
Konec.
```

//-----

7.1.6 N: Funkcija *Fib()* - Fibonaccijevo število

Napišimo funkcijo `int fib(int n)`, ki v *for* zanki izračuna Fibonaccijevo število za število *n*.

Z uporabo te funkcije izpišimo prvih 35 členov Fibonaccijevega zaporedja.

Namen: Izdelava funkcije `int fib(int n)` z vsemi potrebnimi lokalnimi spremenljivkami za računanje v zanki, uporaba te funkcije v stavku izpisa v zanki.

Izpis:

```
f ib<0>=0
f ib<1>=1
f ib<2>=1
f ib<3>=2
f ib<4>=3
f ib<5>=5
f ib<6>=8
f ib<7>=13
f ib<8>=21
f ib<9>=34
f ib<10>=55
f ib<11>=89
f ib<12>=144
f ib<13>=233
f ib<14>=377
f ib<15>=610
f ib<16>=987
f ib<17>=1597
f ib<18>=2584
f ib<19>=4181
f ib<20>=6765
f ib<21>=10946
f ib<22>=17711
f ib<23>=28657
f ib<24>=46368
f ib<25>=75025
f ib<26>=121393
f ib<27>=196418
f ib<28>=317811
f ib<29>=514229
f ib<30>=832040
f ib<31>=1346269
f ib<32>=2178309
f ib<33>=3524578
f ib<34>=5702887
```

//-----

7.1.7 N: Funkcija s poljem

Napišimo funkcijo

`int vsota_v_polju(int polje[], int dolzina_polja)`, ki izračuna in vrne vsoto elementov polja. Funkcijo uporabimo v programu, kjer najprej definiramo dve polji `[1,1,1,1,1,1,1,1,1,1]`, `[1,2,3,4,5]`, potem dvakrat kličemo `vsota_v_polju()` in izpišemo vsoti elementov obeh polj.

Namen: Uporaba polja spremenljivk v parametru funkcije. Kot ločen parameter je potrebno dodati dolžino polja.

Izpis:

```
Polje1, [1,1,1,1,1,1,1,1,1,1], vsota elementov: 10  
Polje2, [1,2,3,4,5], vsota elementov: 15
```

//-----

7.1.8 N: Fibonacci rekurzivno

Izračunajmo prvih 35 elementov Fibonaccijevega zaporedja z uporabo rekurzivne funkcije.

Namen: Izdelava funkcije

`unsigned int fib(unsigned int n)` za rekurzivno rabo. Rekurzivna funkcija kliče sama sebe tolikokrat, kolikorkrat je potrebno za izračun rezultata.

Izpis:

```
f i b ( 0 ) = 0
f i b ( 1 ) = 1
f i b ( 2 ) = 1
f i b ( 3 ) = 2
f i b ( 4 ) = 3
f i b ( 5 ) = 5
f i b ( 6 ) = 8
f i b ( 7 ) = 13
f i b ( 8 ) = 21
f i b ( 9 ) = 34
f i b ( 1 0 ) = 55
f i b ( 1 1 ) = 89
f i b ( 1 2 ) = 144
f i b ( 1 3 ) = 233
f i b ( 1 4 ) = 377
f i b ( 1 5 ) = 610
f i b ( 1 6 ) = 987
f i b ( 1 7 ) = 1597
f i b ( 1 8 ) = 2584
f i b ( 1 9 ) = 4181
f i b ( 2 0 ) = 6765
f i b ( 2 1 ) = 10946
f i b ( 2 2 ) = 17711
f i b ( 2 3 ) = 28657
f i b ( 2 4 ) = 46368
f i b ( 2 5 ) = 75025
f i b ( 2 6 ) = 121393
f i b ( 2 7 ) = 196418
f i b ( 2 8 ) = 317811
f i b ( 2 9 ) = 514229
f i b ( 3 0 ) = 832040
f i b ( 3 1 ) = 1346269
f i b ( 3 2 ) = 2178309
f i b ( 3 3 ) = 3524578
f i b ( 3 4 ) = 5702887
```

//-----

7.1.9 N: Faktoriela rekurzivno

Naredimo program za računanje faktorielle v rekurzivni funkciji `int faktoriela(int n)`. Izpisujemo števec rabe te funkcije ob vsakem klicu.

Namen: Izdelava funkcije `int faktoriela(int n)` za rekurzivno rabo. Rekurzivna funkcija kliče sama sebe tolikokrat, kolikorkrat je potrebno za izračun rezultata. Uporaba statične lokalne spremenljivke *števec*, ki omogoči izpis števila klicev funkcije `int faktoriela(int n)`.

Izpis:

```
Racunanje faktorielle v rekurzivni funkciji.
Upisite stevilo 0<=n<=12, kateremu racunamo n!, 999 za konec:
10
Stevec rabe rekurzivne funkcije faktoriela: 1, argument = 10
Stevec rabe rekurzivne funkcije faktoriela: 2, argument = 9
Stevec rabe rekurzivne funkcije faktoriela: 3, argument = 8
Stevec rabe rekurzivne funkcije faktoriela: 4, argument = 7
Stevec rabe rekurzivne funkcije faktoriela: 5, argument = 6
Stevec rabe rekurzivne funkcije faktoriela: 6, argument = 5
Stevec rabe rekurzivne funkcije faktoriela: 7, argument = 4
Stevec rabe rekurzivne funkcije faktoriela: 8, argument = 3
Stevec rabe rekurzivne funkcije faktoriela: 9, argument = 2
Stevec rabe rekurzivne funkcije faktoriela: 10, argument = 1
Stevec rabe rekurzivne funkcije faktoriela: 11, argument = 0
10!=3628800
Upisite stevilo 0<=n<=12, kateremu racunamo n!, 999 za konec:
999
Konec.
```

//-----

7.2 Rešitve (R)

7.2.1 R: Vsota števil

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
int preberi(void);  
int vsota(int a, int b);  
void izpisi(int a);  
  
int preberi( void )  
{ int w;  
  printf( "Vnesi celo stevilo: " );  
  scanf( "%i.", &w );  
  return w; }  
  
int vsota( int a, int b )  
{ return a+b; }  
  
void izpisi( int a )  
{ printf( "Vsota je %i.", a ); }  
//-----  
#pragma argsused  
int main(int argc, char* argv[]) {  
  int x, y, z;  
  puts("");  
  x = preberi();  
  y = preberi();  
  z = vsota( x, y );  
  izpisi( z );  
  puts("");  
  return 0;  
}  
//-----
```

7.2.2 R: Parameter in spremenljivka

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
void zamenjaj(int a, int b);  
  
void zamenjaj( int a, int b )  
{ int w;  
  w = a;  
  a = b;  
  b = w; }  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
  int x = 1;  
  int y = 2;  
  
  printf( "\n%d, %d\n", x, y );  
  
  zamenjaj( x, y );  
  
  printf( "%d, %d\n", x, y );  
  
  return 0;  
}  
//-----
```

7.2.3 R: Globalni spremenljivki

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
void zamenjaj(void);  
int x;  
int y;  
  
void zamenjaj( void )  
{ int w;  
  w = x;  
  x = y;  
  y = w; }  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
  x = 1;  
  y = 2;  
  
  printf( "\n%d, %d\n", x, y );  
  zamenjaj();  
  
  printf( "%d, %d\n", x, y );  
  
  return 0;  
}  
//-----
```

7.2.4 R: Rešitev kvadratne enačbe

```
// resitev brez uporabe knjižnice math

//-----
#include <stdio.h>
#include <stdbool.h>
#pragma hdrstop
//-----

float absolutnaVrednost (float x)
{
    if ( x < 0 )
        x = -x;
    return x;
}

//-----
float kvadratniKoren (float x)
{
    float priblizek = 1.0;
    float napaka = .0000001;

    while ( absolutnaVrednost((priblizek * priblizek) / x - 1.0) >= napaka )
        priblizek = ( x / priblizek + priblizek ) / 2.0;

    return priblizek;
}

#pragma argsused
//-----
int main(int argc, char* argv[])
{
    float a,b,c;
    float diskriminanta, sprotni;
```

```

float rezultat1, rezultat2;
printf ("\nRacunamo korene kvadratne enacbe:\n\n");
do {
    printf ("\nVnesite koeficiente a, b in c, 0 0 0 za konec:\n");
    scanf ("%f %f %f", &a, &b, &c);
    if (!a && !b && !c) break;
    diskriminanta = b * b - 4 * a * c;

    if (diskriminanta < 0) {
        printf ("b * b - 4 * a * c = %f, kompleksna resitev.\n", diskriminanta);
        continue;
    }

    sprotni = kvadratniKoren(diskriminanta);

    rezultat1 = (-b + sprotni) / (2 * a);
    rezultat2 = (-b - sprotni) / (2 * a);
    printf ("Resitvi za %.4f, %.4f, %.4f sta: %f %f\n", a, b, c, rezultat1, rezultat2);

}
while (true);

printf ("\nKonec.");

return 0;
}
//-----

```

```

// resitev z uporabo knjižnice math:

//-----
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
#pragma hdrstop

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    float a,b,c;
    float diskriminanta, sprotni;
    float rezultat1, rezultat2;
    printf ("\nRacunamo korene kvadratne enacbe:\n\n");
    do {
        printf ("\nVnesite koeficiente a, b in c, 0 0 0 za konec:\n");
        scanf ("%f %f %f", &a, &b, &c);
        if (!a && !b && !c) break;

        diskriminanta = pow(b,2) - 4 * a * c;

        if (diskriminanta < 0){
            printf ("b * b - 4 * a * c = %f, kompleksna resitev.\n", diskriminanta);
            continue;
        }

        sprotni = sqrt(diskriminanta);

        rezultat1 = ( -b + sprotni ) / ( 2 * a );
        rezultat2 = ( -b - sprotni ) / ( 2 * a );
        printf ("Resitvi za %.4f, %.4f, %.4f sta: %f %f\n", a, b, c, rezultat1, rezultat2);

    }
    while (true);

```



```
printf ("\nKonec.");
```

```
return 0;
```

```
}
```

```
//-----
```

7.2.5 R: Funkcija Največji Skupni Delitelj (NSD)

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
int NSD(int, int);  
  
int NSD(int a, int b) {  
    // za npr 10 in 4  
    // a % b = c  
    // 10 % 4 = 2  
    // 4 % 2 = 0  
    // 2 je NSD števil 10 in 4  
  
    int c;  
    if (a == 0 && b == 0) return 0;  
    if (a == 0) return b;  
    if (b == 0) return a;  
    while (b != 0) {  
        c = a%b;  
        a = b;  
        b = c;  
    }  
    return a;  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int a, b;
```

```
puts("\nNajvecji skupni delitelj (NSD) dveh celih stevil.");
```

```
while (true) {  
    printf("\nVnesi 1. stevilo, -1 za konec: ");  
    scanf("%i", &a);  
    if (a == -1) break;  
    printf("Vnesi 2. stevilo, -1 za konec: ");  
    scanf("%d", &b);  
    if (b == -1) break;  
    printf("NSD(%d, %d) = %d\n", a, b, NSD(a, b));  
}
```

```
puts ("\nKonec.");
```

```
return 0;
```

```
}
```

```
//-----
```

7.2.6 R: Funkcija *Fib()* – Fibonaccijevo število

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
int fib (int n)  
{ int f0 = 0, f1 = 1, f, i;  
  if (n == 0) return 0;  
  for (i=0; i < n-1; i++) {  
    f = f1;  
    f1 = f0 + f1;  
    f0 = f;  
  }  
  return f1;  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
  int n;  
  puts("");  
  for(n=0; n<35; ++n) printf("fib(%d)=%d\n", n, fib(n));  
  
  return 0;  
}  
//-----
```

7.2.7 R: Funkcija s poljem

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
int vsota_v_polju(int vrednosti[], int n)  
{  
    int i;  
    int vsota = 0;  
  
    for (i = 0; i < n; ++i)  
        vsota += vrednosti[i];  
  
    return vsota;  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define DIM_1 10  
    #define DIM_2 6  
  
    int i;  
    int Polje1[DIM_1];  
    int Polje2[DIM_2];  
  
    for (i=0; i < DIM_1; i++)  
        Polje1[i]=1;  
    for (i=0; i < DIM_2; i++)  
        Polje2[i]=i;
```

```
    printf ("\nPolje1, [1,1,1,1,1,1,1,1,1], vsota elementov: %i\n",  
vsota_v_polju(Polje1, DIM_1));  
    printf ("Polje2, [1,2,3,4,5], vsota elementov: %i\n", vsota_v_polju(Polje2,  
DIM_2));  
    return 0;  
}  
//-----
```

7.2.8 R: Fibonacci rekurzivno

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h>  
#pragma hdrstop  
  
//-----  
unsigned int fib(unsigned int n)  
{  
    return n < 2 ? n : fib(n-1) + fib(n-2);  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int n;  
    puts("");  
    for(n=0; n<35; ++n) printf("fib(%u)=%u\n", n, fib(n));  
  
    return 0;  
}  
//-----
```

7.2.9 R: Faktoriela rekurzivno

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
int faktoriela(int n);  
int faktoriela(int n) {  
    static int stevec = 0;  
    printf("Stevec rabe rekurzivne funkcije ""faktoriela"": %2i, argument = %2i\n",  
++stevec, n);  
    if (n==0)  
        return 1;  
    else  
        return n * faktoriela(n-1);  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[]) {  
    int stevilo;  
    puts("\nRacunanje faktoriele v rekurzivni funkciji.");  
    while (true) {  
        puts("\nVpisite stevilo 0<=n<=12, kateremu racunamo n!, 999 za konec: ");  
        scanf ("%i", &stevilo);  
        if (stevilo == 999 || stevilo<0 || stevilo>12) break;  
        printf( "\n%i!=%d\n\n", stevilo, faktoriela(stevilo) );  
    }  
    puts("\nKonec.");  
    return 0;  
}  
//-----
```


8 Strukture

Poudarki:

- Definicija podatkovne strukture *struct*.
- V strukturo vgnezdena struktura.
- Podatkovna struktura uporabljena kot parameter funkcije.
- Inicializacija vrednosti v podatkovni strukturi.

8.1 Naloge

8.1.1 N: Razdalja

Določimo strukturo *tocka(x,y)* in strukturo *krog(tocka,r)*.

Program naj izračuna razdaljo med krožnico in točko za

točka: {3, 8}

krog: {{3, 4} 2.5}

Namen: Določitev strukture za koordinato točke in določitev sestavljene strukture za krog – koordinata središča in radij. Uporaba struktur v računu.

Izpis:

```
Racunamo razdaljo med kroznico in tocko.  
Krog(x=3, y=4, r=2.5), tocka(x=3, y=8)  
Razdalja je 1.5 enot.
```

//-----

8.1.2 N: Pretečeni čas

Določimo strukturo *cas(ure, minute, sekunde)*. Izračunajmo pretečene čase med različnimi časi:

od { 3, 45, 15 } do { 9, 44, 03 }

od { 9, 44, 03 } do { 3, 45, 15 }

od {22, 50, 59 } do { 7, 30, 0 }

Pretečene čase računajmo v funkciji

struct sCas pretezeniCas (struct sCas t1, struct sCas t2), ki izračuna $t2 - t1$.

Namen: Računanje s strukturami. Prenos podatkov preko struktur v funkcijo

struct sCas pretezeniCas (struct sCas t1, struct sCas t2);

in preko rezultata v obliki strukture nazaj v glavni program.

Izpis:

```
Cas med 03:45:15 in 09:44:03 : 05:58:48
Cas med 09:44:03 in 03:45:15 : 18:01:12
Cas med 22:50:59 in 07:30:00 : 08:39:01
```

//-----

8.2 Rešitve

8.2.1 R: Razdalja

```
//-----  
#include <stdio.h>  
#include <math.h>  
#pragma hdrstop  
//-----  
  
struct tocka {  
    double x;  
    double y;  
};  
  
struct krog {  
    struct tocka sr;  
    double r;  
};  
  
double razdalja (struct krog k, struct tocka t)  
{  
    double d = sqrt( pow((k.sr.x - t.x),2) + pow((k.sr.y - t.y),2) ) - k.r;  
    if (d < 0.0) d = 0.0;  
    return d;  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    struct tocka tc;  
    struct krog kr;  
  
    tc.x = 3.0;  
    tc.y = 8.0;
```

```
kr.sr.x = 3.0;
kr.sr.y = 4.0;
kr.r = 2.5;
```

```
printf("\nRacunamo razdaljo med kroznico in tocko.\n");
printf("Krog(x=%lg, y=%lg, r=%lg), tocka(x=%lg, y=%lg)\n",
kr.sr.x, kr.sr.y, kr.r, tc.x, tc.y);
```

```
printf( "Razdalja je %lg enot.\n", razdalja(kr, tc));
```

```
return 0;
}
```

```
//-----
```

8.2.2 R: Pretečeni čas

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
struct sCas  
{  
    int ure;  
    int minute;  
    int sekunde;  
};  
  
struct sCas preteцениCas (struct sCas t1, struct sCas t2)  
{  
    struct sCas rezultat = {0, 0, 0};  
    // najprej razliko sekund v sekunde  
    rezultat.sekunde = t2.sekunde - t1.sekunde;  
  
    // ce rezultat.sekunde < 0, vzamemo eno minuto  
    if (rezultat.sekunde < 0) {  
        rezultat.sekunde += 60;  
        --t2.minute; }  
  
    // razliko minut v minute  
    rezultat.minute = t2.minute - t1.minute;  
    // ce rezultat.minute < 0, vzamemo eno uro  
    if (rezultat.minute < 0) {  
        rezultat.minute += 60;  
        --t2.ure; }  
  
    // razliko ur v ure  
    rezultat.ure = t2.ure - t1.ure;  
    // ce rezultat.ure < 0, vzamemo en dan  
    if (rezultat.ure < 0)
```

```

    rezultat.ure += 24;
    return rezultat; }

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    // ure, minute, sekunde
    // casi so t1, t2, t3, t4, rezultat
    struct sCas t1 = { 3, 45, 15 }, t2 = { 9, 44, 03 },
                t3 = {22, 50, 59 }, t4 = { 7, 30, 0 };
    struct sCas rezultat;

    struct sCas preteceeniCas (struct sCas t1, struct sCas t2);

    rezultat = preteceeniCas (t1, t2);
    printf ("\nCas med %.2i:%.2i:%.2i in %.2i:%.2i:%.2i "
           ": %.2i:%.2i:%.2i\n",
           t1.ure, t1.minute, t1.sekunde, t2.ure, t2.minute,
           t2.sekunde, rezultat.ure, rezultat.minute, rezultat.sekunde);

    rezultat = preteceeniCas (t2, t1);
    printf ("Cas med %.2i:%.2i:%.2i in %.2i:%.2i:%.2i "
           ": %.2i:%.2i:%.2i\n",
           t2.ure, t2.minute, t2.sekunde, t1.ure, t1.minute,
           t1.sekunde, rezultat.ure, rezultat.minute, rezultat.sekunde);

    rezultat = preteceeniCas (t3, t4);
    printf ("Cas med %.2i:%.2i:%.2i in %.2i:%.2i:%.2i "
           ": %.2i:%.2i:%.2i\n",
           t3.ure, t3.minute, t3.sekunde, t4.ure, t4.minute,
           t4.sekunde, rezultat.ure, rezultat.minute, rezultat.sekunde);

    return 0;
}
//-----

```

9 Nizi

Poudarki:

- Definicija niza znakov.
- Pisanje in branje znakov v nizu.
- Vnos niza z *gets()* in izpis niza s *puts()*, *printf()*.
- Kopiranje niza.
- Obračanje niza.
- Iskanje znaka v nizu.
- Iskanje podniza v nizu.
- Odstranitev podniza iz niza.
- Zamenjava dela niza z drugim nizom.
- Sestavljanje nizov.
- Vstavitve niza v drug niz.
- Niz v podatkovni strukturi.
- Niz kot parameter funkcije.
- Uporaba funkcij iz knjižnice *string*: *strcpy()*, *strcmp()*, *strlen()*, *isalpha()*, *isupper()*, *islower()*, *isdigit()*, *isspace()*, *strcat()*.

9.1 Naloge

9.1.1 N: Vpis, izpis niza

Napišimo program za vnos in izpis niza.

Namen: Uporaba funkcij *puts()* in *gets()*. Dodelitev potrebnih spominskih mest tipa *char* za hrambo vnešenega niza.

Izpis:

```
Upisite niz (končajte z Enter):  
Danes sije sonce (ni res).  
Upisali ste: "Danes sije sonce (ni res).".
```

//-----

9.1.2 N: Obrni niz

Napišimo program, kjer vpišemo niz, ki se izpiše od zadaj naprej.

Namen: Vpis, izpis niza, obračanje niza z menjavo znakov na mestih n in $(dolžinaNiza - n)$.

Izpis:

```
Upisite besedilo: Stefan kuha kosilo.  
Obrnjen niz: .olisok ahuk nafetS
```

//-----

9.1.3 N: Niz v strukturi

Določimo podatkovno strukturo, ki bo vsebovala niz za ime in celo število za vnos starosti. Program naj vpraša po starosti in naj izpiše ime in starost.

Namen: Določitev podatkovne strukture, ki sestoji iz niza in iz celega števila. Uporaba *strcpy()* kot eno od možnosti za inicializacijo niza.

Izpis:

```
Koliko je star Janez? 59  
Janez je star 59 let.
```

//-----

9.1.4 Naredimo najdiNiz()

Napišimo funkcijo, ki ugotovi, če v nizu obstoja določen podniz. Deklaracija funkcije:

```
int najdiNiz (const char glavniNiz[], const char podNiz[]);
```

Uporaba funkcije:

```
index = najdiNiz («Zelena dolina«, »dolina«);
```

V danem primeru funkcija vrne 7 – mesto prve črke iskanega niza, začenši s pozicijo 0. Če se iskanega podniza ne najde, naj funkcija vrne -1.

Namen: Delo z nizom na nivoju *char* spremenljivk, dolžine niza, znaka '\0' ki določa konec niza.

Izpis:

```
Iscemo indeks niza "sinonim" v nizu "Zelena dolina je sinonim za bio hrano."  
Prvi znak 'Z' ima indeks 0  
Indeks = 17.
```

//-----

9.1.5 N: Naredimo *odstraniNiz()*

Napišimo funkcijo, ki iz niza odstrani določeno število znakov od določenega mesta naprej. Deklaracija funkcije:

```
bool odstraniNiz (char Niz[], int ZacetekVen, int SteviloVen);
```

Uporaba funkcije:

```
bool odstraniNiz ("Sosed ima drag restavriran rdeč avto.", 15, 12)
```

V danem primeru je rezultat " Sosed ima drag rdeč avto. "

Funkcija naj vrne *false*, če katerikoli od indeksov odstranjenih črk presega dolžino niza.

Namen: Delo z nizom na nivoju znakov.

Izpis:

```
Zacetni niz:
Sosed ima drag restavriran rdec avto.
Po funkciji bool odstraniNiz (<"Sosed ima drag restavriran rdec avto.", 15, 12>;
Sosed ima drag rdec avto.
```

//-----

9.1.6 N: Naredimo *vstaviNiz()*

Napišimo funkcijo, ki v niz vrine drug niz. Deklaracija funkcije:

```
bool vstaviNiz (char glavniNiz[], const char podNiz[], int index);
```

Uporaba funkcije:

```
vstaviNiz ("drag avto", "restavriran", 5);
```

Nastane niz "drag restavriran avto".

Funkcija vrne *false*, če vstavljamo drugi niz z indeksom večjim od dolžine prvega niza.

Namen: Delo z nizom na nivoju znakov.

Izpis:

```
Zacetni niz:
Sosed ima avto.
Po funkciji bool vstaviNiz ("Sosed ima avto.", "drag restavriran rdec ", 10);
Sosed ima drag restavriran rdec avto.
```

//-----

9.1.7 N: Naredimo *zamenjajNiz()*

Napišimo funkcijo, ki del niza zamenja z drugim nizom.

Deklaracija funkcije:

```
void zamenjajNiz (char nizGlavni[], const char nizOdstrani[], const  
char niz Vstavi[])
```

Uporaba funkcije:

```
zamenjajNiz(text, "1", "ena");
```

zamenja prvo "1" z "ena" v nizu *text*, če "1" v nizu *text* obstoja.

Pri izdelavi funkcije *ZamenjajNiz* si lahko pomagamo z že narejenimi funkcijami *najdiNiz*, *odstraniNiz*, *vstaviNiz*.

Namen: Uporaba že narejenih funkcij pri izdelavi nove funkcije, da ne podvajamo že opravljenega dela.

Izpis:

```
Predsednik na leto dobi 120 000 eur denarja.  
Predsednik na leto dobi stodvajset tisoc eur denarja.  
Predsednik na leto dobi stodvajset tisoc eurov denarja.  
  
Pisimo *** na tem mestu.  
Pisimo ZUEZDICE na tem mestu.
```

//-----

9.1.8 N: Preštej besede

Napišimo program, ki prebere stavke in prešteje, koliko je besed v njem. Beseda je poljubno zaporedje znakov, posamezne besede pa so ločene z enim ali več presledki. Presledki se lahko pojavljajo tudi na začetku ali na koncu stavka.

Vpišite niz: npr. "Mount Everest je visok 8846 metrov."

V nizu je 6 besed.

Namen: Delo z nizom na nivoju znakov. Upoštevanje, da se niz lahko konča z alfanumericnim znakom ali s presledkom.

Izpis:

```
Upisite niz: Mount Everest je visok 8846 metrov.  
V nizu je 6 besed.
```

//-----

9.1.9 N: Sprememba velikosti črk

Napišimo program, ki bo v danem nizu vse velike črke pretvoril v ustrezne male črke, in obratno. Vse druge znake naj pusti nespremenjene. Uporabimo funkcije iz knjižnice prevajalnika.

Namen: Uporaba funkcij *strlen()*, *isupper()*, *islower()*, *toupper()*, *tolower()*.

Izpis:

```
Upisite niz: UELIKE in male CrKe.  
Spremenjen niz: velike IN MALE crke.
```

//-----

9.1.10 N: Palindrom?

Napišimo program, ki ugotovi, ali je vnešeno besedilo palindrom. (pomeni: enako se bere od zadaj naprej kot od spredaj nazaj.)

Namen: Najprej odstranitev znakov, ki niso alfanumerični, sledi pretvorba preostalih znakov v velike ali male črke, obrnitev niza in ugotavljanje enakosti filtriranega in obrnjenega niza.

Izpis:

```
Ali vpisemo palindrom (vpis in obrnjen vpis sta enaka)?
Upisite besedilo, "konec" za konec: ana ana
Besedilo je palindrom.
Upisite besedilo, "konec" za konec: ana bolana
Besedilo ni palindrom.
Upisite besedilo, "konec" za konec: konec
Konec.
```

//-----

9.1.11 N: Značilke niza

Napišimo program, ki karakterizira lastnosti vnešenega niza: dolžino, število črk, število velikih črk, število malih črk, število števčk, število presledkov.

Namen: Uporaba funkcij *strlen()*, *isalpha()*, *isupper()*, *islower()*, *isdigit()*, *isspace()*.

Izpis:

```
Upisite niz, "konec" za konec: Podjetje "Danone" je kupilo reklamo za 20000 eur.
Dolžina niza:          49
Število crk:          34
Število velikih crk:  2
Število malih crk:    32
Število stevk:        5
Število presledkov:   7

Upisite niz, "konec" za konec: konec
Konec.
```

//-----

9.1.12 N: Združitev nizov

Napišimo program, ki združi dva vpisana niza.

Namen: Uporaba funkcije *strcat()*.

Izpis:

```
Upisite prvi niz: Ta okna  
Upisite drugi niz: izolirajo odlicno.  
Zdruzeni niz: Ta oknaizolirajo odlicno.
```

//-----

9.2 Rešitve

9.2.1 R: Vpis, izpis niza

```
//-----  
#include <stdio.h>  
#include <string.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    char niz[256];  
    puts("\nVpisite niz (koncaj z Enter):");  
    gets(niz);  
    printf("Vpisali ste: \"%s\".\n",niz);  
  
    return 0;  
}  
//-----
```


9.2.2 R: Obrni niz

```
//-----  
#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
void obrni(char []);  
  
void obrni(char s[]) {  
    int i, j;  
    int n = strlen(s);  
  
    for (i=0, j=n-1; i<j; i++, j--) {  
        char c = s[i];  
        s[i] = s[j];  
        s[j] = c;  
    }  
}  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    char niz[200];  
    printf("\nVpisite besedilo: ");  
    gets(niz);  
  
    obrni(niz);  
    printf("Obrnjen niz: %s\n", niz);  
    return 0;  
}  
  
//-----
```

9.2.3 R: Niz v strukturi

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
struct oseba {  
    char ime[20];  
    int starost;  
};  
  
#pragma argsused  
//-----  
int main(int argc, char* argv[])  
{  
    strcpy( x.ime,"Janez");  
    printf("\nKoliko je star %s? ",x.ime);  
    scanf("%d",&x.starost);  
    printf("%s je star %d let.",x.ime, x.starost);  
  
    return 0;  
}  
  
//-----
```

9.2.4 R: Naredimo najdiNiz()

```
//-----  
#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
int najdiNiz (const char glavniNiz[], const char podNiz[]);  
  
int najdiNiz (const char glavniNiz[], const char podNiz[]) {  
    int i, j, naseNiz = false;  
  
    for ( i = 0; glavniNiz[i] != '\0' && !naseNiz; ++i ) {  
        naseNiz = true;  
        for ( j = 0; podNiz[j] != '\0' && naseNiz; ++j )  
            if ( glavniNiz[j + i] != podNiz[j] || glavniNiz[j + i] == '\0' )  
                naseNiz = false;  
            if (naseNiz)  
                return i; // indeks zacetka podNiza  
    }  
    return -1;  
}  
//-----  
#pragma argsused  
int main(int argc, char* argv[]) {  
    int index = najdiNiz ("Zelena dolina je sinonim za bio hrano.", "sinonim");  
  
    puts ("\nIscemo indeks niza \"sinonim\" v nizu \"Zelena dolina je sinonim za  
bio hrano.\"");  
    puts ("Prvi znak 'Z' ima indeks 0");  
    printf ("Indeks = %i.\n", index);  
  
    return 0;  
}  
//-----
```

9.2.5 R: Naredimo *odstraniNiz()*

```
//-----  
#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
bool odstraniNiz (char Niz[], int ZacetekVen, int SteviloVen)  
{  
    int i, NizDolzina;  
    for (i=0; true; i++)  
        if (Niz[i]=='\0') {  
            NizDolzina = i;  
            break;  
        }  
    if (NizDolzina < (ZacetekVen + SteviloVen-1)) return false;  
  
    for (i = ZacetekVen; Niz[i] != '\0'; i++)  
        Niz[i] = Niz[i + SteviloVen];  
  
    Niz[i] = '\0';  
    return true;  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    char Niz[] = "Sosed ima drag restavriran rdec avto."  
  
    puts("\nZacetni niz:");  
    printf ("%s\n", Niz);  
    printf ("%s", "\nPo funkciji bool odstraniNiz (\nSosed ima drag restavriran  
rdec avto.\n", 15, 12);\n\n");
```

```
odstraniNiz(Niz, 15, 12);
```

```
printf ("%s\n", Niz);
```

```
return 0;
```

```
}
```

```
//-----
```

9.2.6 R: Naredimo *vstaviNiz()*

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
int dolzinaNiza (const char niz[])  
{  
    int i = 0;  
    while ( niz[i] != '\0' )  
        ++i;  
    return i;  
}  
  
//-----  
bool vstaviNiz (char glavniNiz[], const char podNiz[], int index)  
{  
    int j, glavniNizDolzina, podNizDolzina;  
  
    glavniNizDolzina = dolzinaNiza (glavniNiz);  
    podNizDolzina = dolzinaNiza (podNiz);  
  
    if (index > glavniNizDolzina)  
        return false;  
  
    for ( j = glavniNizDolzina; j >= index; --j )  
        glavniNiz[podNizDolzina + j] = glavniNiz[j];  
  
    // prostor narejen, kopiranje v ta prostor  
    for ( j = 0; j < podNizDolzina; ++j )  
        glavniNiz[j + index] = podNiz[j];  
  
    return true;  
}
```

```

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    char Niz[60] = "Sosed ima avto."; // !, potrebna razerva alociranega prostora
        // novi niz bo daljsi
    char podNiz[] = "drag restavriran rdec ";

    puts("\nZacetni niz:");
    printf ("%s\n", Niz);

    printf ("%s", "\nPo funkciji bool vstaviNiz (\\"Sosed ima avto.\", \\"drag
restavriran rdec \", 10);\n\n");
    vstaviNiz (Niz, podNiz, 10);

    printf ("%s\n", Niz);

    return 0;
}

//-----

```

9.2.7 R: Naredimo *zamenjajNiz()*

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
int dolzinaNiza (const char niz[])  
{  
    int i = 0;  
    while ( niz[i] != '\0' )  
        ++i;  
    return i;  
}  
//-----  
bool vstaviNiz (char glavniNiz[], const char podNiz[], int index)  
{  
    int j, glavniNizDolzina, podNizDolzina;  
  
    glavniNizDolzina = dolzinaNiza (glavniNiz);  
    podNizDolzina = dolzinaNiza (podNiz);  
  
    if (index > glavniNizDolzina)  
        return false;  
  
    for ( j = glavniNizDolzina; j >= index; --j )  
        glavniNiz[podNizDolzina + j] = glavniNiz[j];  
  
    // prostor narejen, kopiranje v ta prostor  
    for ( j = 0; j < podNizDolzina; ++j )  
        glavniNiz[j + index] = podNiz[j];  
  
    return true;  
}
```



```

//-----
bool odstraniNiz (char Niz[], int ZacetekVen, int SteviloVen)
{
    int i, NizDolzina;
    for (i=0; true; i++)
        if (Niz[i]=='\0') {
            NizDolzina = i;
            break;
        }
    if (NizDolzina < (ZacetekVen + SteviloVen-1)) return false;

    for (i = ZacetekVen; Niz[i] != '\0'; i++)
        Niz[i] = Niz[i + SteviloVen];

    Niz[i] = '\0';
    return true;
}

//-----
int najdiNiz (const char glavniNiz[], const char podNiz[])
{
    int i, j, naseINiz = false;

    for ( i = 0; glavniNiz[i] != '\0' && !naseINiz; ++i ) {
        naseINiz = true;
        for ( j = 0; podNiz[j] != '\0' && naseINiz; ++j )
            if ( glavniNiz[j + i] != podNiz[j] || glavniNiz[j + i] == '\0' )
                naseINiz = false;
            if (naseINiz)
                return i; // indeks zacetka podNiza
    }
    return -1;
}

//-----
void zamenjajNiz (char nizGlavni[], const char nizOdstrani[], const char
nizVstavi[])

```

```

{
    int najdiNiz (const char glavniNiz[], const char podNiz[]);
    int dolzinaNiza (const char niz[]);
    bool odstraniNiz (char Niz[], int ZacetekVen, int SteviloVen);
    bool vstaviNiz (char glavniNiz[], const char podNiz[], int index);

    int iPozicija = najdiNiz(nizGlavni, nizOdstrani);
    int iDolzina = dolzinaNiza(nizOdstrani);
    if (odstraniNiz(nizGlavni, iPozicija, iDolzina)) {
        vstaviNiz(nizGlavni, nizVstavi, iPozicija); }
}

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    char nizZacetni[60] = "Predsednik na leto dobi 120 000 eur denarja.";
    char nizDrugi[60] = "Pisimo *** na tem mestu.";

    printf ("\n%s\n", nizZacetni);
    zamenjajNiz (nizZacetni, "120 000", "stodvajset tisoc");
    printf ("%s\n", nizZacetni);
    zamenjajNiz (nizZacetni, "eur", "eurov");
    printf ("%s\n", nizZacetni);

    printf ("\n%s\n", nizDrugi);
    zamenjajNiz (nizDrugi, "***", "ZVEZDICE");
    printf ("%s\n", nizDrugi);

    return 0;
}
//-----

```

9.2.8 R: Preštej besede

```
//-----  
#include <stdio.h>  
#include <string.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define MAX 200  
  
    char niz[MAX];  
    int stevec, len, i;  
  
    printf("\nVpisite niz: ");  
    gets(niz);  
  
    stevec = 0;  
    for (i=0; niz[i] != '\0'; i++) {  
        if (niz[i] != ' ' && niz[i+1] == ' ' )  
            stevec++;  
    }  
    if (niz[strlen(niz)-1]==' ') stevec--; // situacija " aa bb cc "  
  
    printf("V nizu je %d besed.\n", ++stevec); // dodana zadnja beseda  
  
    return 0;  
}  
//-----
```

9.2.9 R: Sprememba velikosti črk

```
//-----  
#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
#include <ctype.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define MAX 200  
    char niz[MAX];  
    unsigned int i;  
  
    printf("\nVpisite niz: ");  
    gets(niz);  
  
    for (i=0; i<strlen(niz); i++)  
        if (islower(niz[i]))  
            niz[i]=toupper(niz[i]);  
        else if (isupper(niz[i]))  
            niz[i]=tolower(niz[i]);  
  
    printf("Spremenjen niz: %s\n", niz);  
  
    return 0;  
}  
//-----
```

9.2.10 R: Palindrom?

```
//-----  
#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
#include <ctype.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    char niz[200], nizObrnjeni[200];  
    int len, i, j;  
  
    puts("\nAli vpisemo palindrom (vpis in obrnjen vpis sta enaka)?\n");  
    while (true) {  
        printf("Vpisite besedilo, \"konec\" za konec: ");  
        gets(niz);  
        if(strcmp(niz,"konec")==0) break;  
  
        // odstranimo vse znake, ki niso crke  
        for (i=0, j=0; niz[i]!='\0'; i++) {  
            if (isalpha(niz[i])) {  
                niz[j] = niz[i];  
                j++;  
            }  
        }  
        niz[j] = '\0';  
  
        // vse velike crke spremenimo v male  
        for (i=0; niz[i] != '\0'; i++) {  
            niz[i]=tolower(niz[i]);  
        }  
    }  
}
```

```

// obrnemo niz, vrednost i od zgoraj
len = i;
for (i=0; i<len; i++) {
    nizObrnjeni[i] = niz[len-i-1];
}
nizObrnjeni[i] = '\0';

if (strcmp(niz, nizObrnjeni) == 0) {
    printf("Besedilo je palindrom.\n\n");
} else {
    printf("Besedilo ni palindrom.\n\n");
}
}
puts("\nKonec.");

return 0;
}

//-----

```

9.2.11 R: Značilke niza

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <string.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    char niz[80];  
    unsigned int i = 0;  
    int crke, crkeVelike, crkeMale, stevke, presledki;  
    crke = crkeVelike = crkeMale = stevke = presledki = 0;  
  
    while (true) {  
        printf("\nVpisite niz, \"konec\" za konec: ");  
        gets(niz);  
        if (0==strcmp(niz,"konec")) break;  
  
        for (i=0; i<strlen(niz); i++) {  
            if (isalpha(niz[i])) crke++;  
            if (isupper(niz[i])) crkeVelike++;  
            if (islower(niz[i])) crkeMale++;  
            if (isdigit(niz[i])) stevke++;  
            if (isspace(niz[i])) presledki++;  
        }  
  
        printf("Dolzina niza:      %d\n", strlen(niz));  
        printf("Stevilo crk:           %d\n", crke);  
        printf("Stevilo velikih crk: %d\n", crkeVelike);  
        printf("Stevilo malih crk:   %d\n", crkeMale);  
        printf("Stevilo stevk:       %d\n", stevke);  
        printf("Stevilo presledkov:  %d\n\n", presledki);
```

```
}  
puts("Konec.");  
  
return 0;  
}  
//-----
```


9.2.12 R: Združitev nizov

```
//-----  
#include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define DIM 80  
    char niz1[DIM];  
    char niz2[DIM];  
  
    printf("\nVpisite prvi niz: ");  
    gets(niz1);  
    printf("Vpisite drugi niz: ");  
    gets(niz2);  
  
    strcat(niz1, niz2);  
  
    printf("Zdruzeni niz: %s", niz1 );  
  
    return 0;  
}  
//-----
```

10 Kazalci

Poudarki:

- Uporaba usmeritvenega operatorja * (*angl. indirection operator*) za definiranje kazalca na spremenljivko.
- Uporaba naslovnega operatorja & (*angl. address operator*) za ugotovitev spominskega naslova spremenljivke.
- Dinamična (sprotna) dodelitev spomina z *malloc()* iz knjižnice *stdlib*.
- Sprostitev dinamično dodeljenega spomina s *free()* iz knjižnice *stdlib*.
- Dinamična (sprotna) izdelava povezanega seznama (*angl. linked list*).
- Uporaba kazalcev v aritmetičnih izrazih.
- Uporaba kazalca za parameter funkcije - funkcija vrača vrednost preko kazalca.
- Uporaba kazalca na funkcijo.

10.1 Naloge

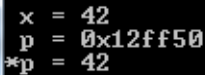
10.1.1 N: Kazalci, *

Pojasnite izpis spodnjih treh *printf()* stavkov:

```
int main(int argc, char* argv[])
{
    int x, *p;
    x = 42;
    p = &x;
    printf(" x = %d\n", x);
    printf(" p = 0x%x\n", p);
    printf("*p = %d\n", *p);
    return 0;
}
```

Namen: Razumevanje delovanja usmeritvenega operatorja (*angl. indirection operator*) *. Razlikovanje med spominsko lokacijo in zapisom spremenljivke na spominski lokaciji.

Izpis:



```
x = 42
p = 0x12ff50
*p = 42
```

//-----

10.1.2 N: Kazalci, * in &

Pojasnite izpise x, y, *p:

```
int main(int argc, char* argv[])
{
    int x, y, *p;
    x = 42;
    y = x;
    p = &x;
    printf("x=%d, y=%d, *p=%d\n", x, y, *p);
    x = 24;
    printf("x=%d, y=%d, *p=%d\n", x, y, *p);
    return 0;
}
```

Namen: Razumevanje delovanja naslovnega operatorja (*angl. address operator*) &. Razlikovanje med kazalcem na spominsko lokacijo in med zapisom vrednosti na tej spominski lokaciji.

Izpis:

```
x=42, y=42, *p=42
x=24, y=42, *p=24
```

//-----

10.1.3 N: Kazalci, dinamična dodelitev spremenljivke

Pojasnite delovanje kode:

```
int main(int argc, char* argv[])
{
    int *p;
    p = malloc( sizeof(int) );
    *p = 42;
    printf(" *p=%d\n", *p);
    free( p );
    return 0;
}
```

Namen: Dinamična dodelitev spremenljivke, vpis, izpis in dinamičen izbris te spremenljivke.

Izpis:

A terminal window showing the output of the program, which is the text '*p=42'.

//-----

10.1.4 N: Kazalec in operatorji *, &, ++

Pojasnite delovanje kode:

```
int main(int argc, char* argv[])
{
    int a[]={0, 11, 22, 33};
    int *p, x;

    p = &a[0];
    x = *p++;
    printf( "x=%d\n", x);

    x = *++p;
    printf( "x=%d\n", x);

    x = ++*p;
    printf( "x=%d\n", x);

    return 0;
}
```

Namen: Razumevanje zvez med operatorji *, & in ++.

Izpis:



```
x=0
x=22
x=23
```

//-----

10.1.5 N: Dinamično narejen povezan seznam

Sestavimo povezan seznam s sprotnim dodeljevanjem elementov seznama.

Element seznama:

```
struct sElementSeznama {
    int vrednost;
    struct sElementSeznama *predhodnik;
};
```

Seznam naj ima 10 elementov. Vrednosti elementom dodeljujemo v zanki. Potem jih izpišemo od zadnjega do prvega. Po izpisu seznam dinamično izbrišemo.

Namen: Dinamična postavitve podatkovne strukture. Vrstni red podatkov v seznamu je določen z vrednostjo kazalca v vsakem elementu. Kazalec elementa kaže na spominsko lokacijo prejšnjega elementa. V tako zasnovanem seznamu lahko brišemo podatke in hkrati sproščamo spominske lokacije, lahko vrinemo nove podatke med obstoječe, lahko podaljšujemo podatkovne strukture in ves čas zasedamo samo toliko spominskih lokacij, kolikor je dejansko potrebno za shranjevanje konkretnih podatkov.

Izpis:

```
Dinamicno narejen povezan seznam desetih elementov, vpisujte jim vrednosti.
Element z indeksom 1, vpisite celostevilsko vrednost: 110
Element z indeksom 2, vpisite celostevilsko vrednost: 109
Element z indeksom 3, vpisite celostevilsko vrednost: 108
Element z indeksom 4, vpisite celostevilsko vrednost: 107
Element z indeksom 5, vpisite celostevilsko vrednost: 106
Element z indeksom 6, vpisite celostevilsko vrednost: 105
Element z indeksom 7, vpisite celostevilsko vrednost: 104
Element z indeksom 8, vpisite celostevilsko vrednost: 103
Element z indeksom 9, vpisite celostevilsko vrednost: 102
Element z indeksom 10, vpisite celostevilsko vrednost: 101

Izpis:
Indeks: 10, vrednost: 101
Indeks: 9, vrednost: 102
Indeks: 8, vrednost: 103
Indeks: 7, vrednost: 104
Indeks: 6, vrednost: 105
Indeks: 5, vrednost: 106
Indeks: 4, vrednost: 107
Indeks: 3, vrednost: 108
Indeks: 2, vrednost: 109
Indeks: 1, vrednost: 110
```

//-----

10.1.6 N: Kazalci – parametri funkcije; statična spremenljivka

Napišimo funkcijo in preverimo njeno delovanje:

```
void zamenjaj(int *a, int *b);
```

Ob preverjanju delovanja izpisujemo sprotni števec rabe funkcije.

Namen: Uporaba kazalcev kot parametrov funkcije. Vrednosti kazalca v funkciji ne spreminjamo, lahko pa spreminjamo vrednost na lokaciji, na katero kaže kazalec. Na ta način preko parametrov funkcije lahko spravljamo vrednost spremenljivke v obe smeri – v funkcijo iz funkcije. Brez naslavljanja spremenljivk s kazalci to ni možno. Tega nekoliko nestandardnega načina prenašanja parametrov funkcije ne uporabljamo brez potrebe. Uporaba kazalcev da veliko novih možnosti – smiselno pa je najprej uporabiti osnovne možnosti.

Statično spremenljivko v funkciji smo srečali že v rekurzivni funkciji za računanje faktorielle. Tokrat jo uporabimo v običajni funkciji za štetje klicev te funkcije.

Izpis:

```
Parametra funkcije "zamenjaj()" sta kazalca.  
x = 1, y = 2  
Funkcija "zamenjaj" uporabljena 1.  
Po zamenjaj(), x = 2, y = 1  
Funkcija "zamenjaj" uporabljena 2.  
Po zamenjaj(), x = 1, y = 2
```

//-----

10.1.7 N: Kazalci – parametri funkcij

Napišimo funkcijo z deklaracijo

```
void sortiraj3(int *piA, int *piB, int *piC);
```

ki sortira tri števila tipa *int* v naraščajoče zaporedje. Funkcija naj uporablja

```
void zamenjaj(int *a, int *b);
```

iz prejšnje naloge. Z *main()* preverimo delovanje sortiranja spremenljivk.

Namen: Uporaba kazalcev kot parametrov funkcije.

Izpis:

```
Tri cela stevila za sortiranje v naraščajoče zaporedje, 9 9 9 za konec: 1234 563
33 12399
Tri cela stevila po sortiranju v naraščajoče zaporedje: 1234, 12399, 56333

Tri cela stevila za sortiranje v naraščajoče zaporedje, 9 9 9 za konec: 2 6 9
Tri cela stevila po sortiranju v naraščajoče zaporedje: 2, 6, 9

Tri cela stevila za sortiranje v naraščajoče zaporedje, 9 9 9 za konec: 9 9 9
Konec.
```

//-----

10.1.8 N: Kazalec na funkcijo

Napišimo funkcije:

```
int seštevanje(int x, int y);  
int odštevanje(int x, int y);  
int množenje(int x, int y);  
int deljenje(int x, int y) {
```

in določimo kazalec na funkcijo

```
int (*pF) (int x, int y);
```

Zgornje štiri funkcije kličemo s klicem gornjega kazalca, npr.

```
pf = množenje;
```

in potem kot

```
printf("zmnožek(%d,%d)=%d\n", a, b, pf(a, b));
```

S temi informacijami naredimo kalkulator za celoštevilске gornje operacije.

Namen: Uporaba kazalca na funkcijo.

Izpis:

```
Unesite prvo celostevilsko stevilo, 999 za konec: 456  
Unesite drugo celostevilsko stevilo: 78  
Unesite racunsko operacijo ( +, -, *, / ): +  
vsota(456,78) = 534  
  
Unesite prvo celostevilsko stevilo, 999 za konec: 45  
Unesite drugo celostevilsko stevilo: 22  
Unesite racunsko operacijo ( +, -, *, / ): /  
kvocient(45,22) = 2  
  
Unesite prvo celostevilsko stevilo, 999 za konec: 999  
Konec.
```

//-----

10.2 Rešitve

10.2.1 R: Kazalci, *

p je kazalec na spremenljivko tipa *integer*. Priredimo mu lokacijo spremenljivke x . Tretji *printf* stavek izpiše vrednost na lokaciji p , drugi *printf* stavek pa izpiše vrednost lokacije p .

```
//-----
```

10.2.2 R: Kazalci, * in &

p je kazalec na spremenljivko tipa *integer*. Priredimo mu lokacijo spremenljivke x . S prvim *printf* stavkom izpišemo $*p$, torej p z usmeritvenim operatorjem, kar je *integer* vrednost na lokaciji spremenljivke x , torej vrednost spremenljivke x . Potem x -u priredimo vrednost 24.

Drugi *printf* posledično izpiše za $*p$ vrednost 24.

```
//-----
```

10.2.3 R: Kazalci, dinamična dodelitev spremenljivke

Bistvo dinamičnega kreiranja spremenljivk je, da tekom izvajanja programa dodeljujemo in odvezujemo spominske lokacije po trenutnih potrebah. Na ta način v povprečju programi potrebujejo manj spominskega prostora kot v primeru, ko vse spominske lokacije (običajno predimenzionirano) dodelimo vnaprej, torej takrat, ko program zaženemo.

p je kazalec na spremenljivko tipa *integer*. Iz nabora prostih spominskih lokacij dodelimo 4 byte in določimo, da ima p vrednost te spominske lokacije (funkcija *malloc*). Prek usmeritvenega operatorja $*$ na to lokacijo zapišemo število 42. S *printf* to število preberemo. Na koncu s *free()* spominsko lokacijo sprostimo.

```
//-----
```

10.2.4 R: Kazalec in operatorji *, &, ++

p je kazalec na spremenljivko tipa *integer*. Dodelimo mu naslov prvega elementa polja a . Vrednost tega elementa je 0. Prek usmeritvenega operatorja $*$ to vrednost dodelimo x -u, preden povečamo spominski naslov p -ja za 1. S prvim *printf()* izpišemo 0.

Potem najprej p povečamo za 1 (zdaj kaže na $a[2]$, najprej je kazal na $a[0]$, nato na $a[1]$). Prek usmeritvenega operatorja $*$ to vrednost dodelimo x -u.

Z drugim *printf()* izpišemo vrednost $a[2]$, torej 22.

Potem za 1 povečamo vrednost, na katero kaže p , torej za 1 povečamo vrednost $a[2]$.

S tretjim *printf()* izpišemo 23.

```
//-----
```

10.2.5 R: Dinamično narejen povezan seznam

```
//-----  
#include <stdio.h>  
#include <stdlib.h>  
#pragma hdrstop  
//-----  
  
struct sElementSeznam {  
    int vrednost;  
    struct sElementSeznam *predhodnik;  
};  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    struct sElementSeznam *pTrenutni, *pPrejsnji;  
    int i, j;  
  
    pPrejsnji = NULL;  
  
    puts ("\nDinamicno narejen povezan seznam desetih elementov, vpisujte jim  
vrednosti.\n");  
  
    for(i=1;i<=10;i++) {  
        pTrenutni = (struct sElementSeznam *)malloc(sizeof(struct  
sElementSeznam));  
  
        printf ("Element z indeksom %2i, vpisite celostevilsko vrednost: ", i);  
        scanf("%i", &j);  
        pTrenutni->vrednost = j;  
  
        pTrenutni->predhodnik = pPrejsnji;  
        pPrejsnji = pTrenutni;  
    }  
}
```

```
puts ("\nlzpis:");
while(pTrenutni) {
    printf("Indeks: %2i, vrednost: %2i\n", --i, pTrenutni->vrednost);
    pTrenutni = pTrenutni->predhodnik;
}

return 0;
}
//-----
```

10.2.6 R: Kazalci - parametri funkcij; statična spremenljivka

```
//-----  
#include <stdio.h>  
#include <stdlib.h>  
#pragma hdrstop  
//-----  
void zamenjaj(int *a, int *b);  
void zamenjaj( int *a, int *b ) {  
    static int n = 0;  
    int w;  
    printf( "\nFunkcija \"zamenjaj\" uporabljena %d.\n", ++n );  
  
    w = *a;  
    *a = *b;  
    *b = w;  
}  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int x = 1;  
    int y = 2;  
  
    puts("\parametra funkcije \"zamenjaj()\" sta kazalca.");  
    printf( "x = %i, y = %i\n", x, y );  
    zamenjaj( &x, &y );  
  
    printf( "Po zamenjaj(), x = %i, y = %i\n", x, y );  
    zamenjaj( &x, &y );  
  
    printf( "Po zamenjaj(), x = %d, y = %d\n", x, y );  
    return 0;  
}  
//-----
```

10.2.7 R: Kazalci – parametri funkcij

```
//-----  
#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
void zamenjaj (int *piA, int *piB)  
{  
    int iAux;  
    iAux = *piA;  
    *piA = *piB;  
    *piB = iAux;  
}  
  
void sortiraj3(int *piA, int *piB, int *piC)  
{  
    void zamenjaj (int *piA, int *piB);  
    if (*piB > *piC) zamenjaj(piB, piC);  
    if (*piA > *piB) zamenjaj(piA, piB);  
    if (*piB > *piC) zamenjaj(piB, piC);  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int iA, iB, iC;  
    int *piA = &iA, *piB = &iB, *piC = &iC;  
    void sortiraj3(int *piA, int *piB, int *piC);  
  
    while (true) {  
        printf ("\nTri cela stevila za sortiranje v narascajoco zaporedje, 9 9 9 za konec:  
");
```



```
scanf ("%d %d %d", &iA, &iB, &iC);
if (iA==9 && iB==9 && iC==9) break;

sortiraj3(piA, piB, piC);

printf ("Tri cela stevila po sortiranju v narascajoce zaporedje: %i, %i, %i \n\n",
iA, iB, iC);
}

puts("Konec.");
return 0;
}

//-----
```

10.2.8 R: Kazalec na funkcijo

```
//-----  
#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
int sestevanje(int x, int y) {  
    return x + y;  
}  
  
int odstevanje(int x, int y) {  
    return x - y;  
}  
  
int mnozenje(int x, int y) {  
    return x * y;  
}  
  
int deljenje(int x, int y) {  
    return x / y;  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int a;  
    int b;  
    char op;  
    int (*pf)(int, int);  
    int err = 0;  
  
    while (true) {  
        printf("\nVnesite prvo celostevilsko stevilo, 999 za konec: ");
```

```

scanf("%d", &a);
if (a==999) break;
printf("Vnesite drugo celostevilsko stevilo: ");
scanf("%d", &b);
printf("Vnesite racunsko operacijo ( +, -, *, / ): ");
scanf("%*c%c", &op);

switch (op) {
    case '+': pf=sestevanje; printf("vsota(%i,%i) = %d\n", a, b, pf(a, b)); break;
    case '-': pf=odstevanje; printf("razlika(%i,%i) = %d\n", a, b, pf(a, b)); break;
    case '*': pf=mnozenje; printf("zmnozek(%d,%d) = %d\n", a, b, pf(a, b));
break;
    case '/': if (b==0) err=1; else pf=deljenje; printf("kvocient(%d,%d) = %d\n", a,
b, pf(a, b)); break;
    default: err = 2; break;
}
}

puts("Konec.");
return err;
}
//-----

```

11 Delo z biti

Poudarki:

- Bitni operatorji: bitni *AND* `&`, bitni *OR* `|`, bitni *XOR* `^`, pomik levo `<<`, pomik desno `>>`, bitna negacija `~`.
- Ločevanje med prireditvenimi bitnimi operacijami `&=`, `|=`, `^=`, `<<=`, `>>=` in med aritmetičnimi operacijami `+=`, `-=`, `*=`, `/=`, `%=`.
- Izpis števil v binarni obliki.
- Z bitnim pomikom ugotovljena velikost istih tipov spremenljivk na različnih platformah.
- Iskanje bitnega vzorca v številu.

11.1 Naloge

11.1.1 N: Bitni operatorji

Z 8 bitnima številoma 22 in 3 ugotovimo rezultate bitnih operacij s funkcijami:

`&` `|` `^` `<<` `>>` `~`.

(bitni *AND*, bitni *OR*, bitni *XOR*, pomik levo, pomik desno, bitna negacija)

Namen: Funkcionalnost bitnih operatorjev. Narediti izpis spremenljivk po bitih.

Izpis:

```
x =      0001 0110
y =      0000 0011

x & y =  0000 0010
x | y =  0001 0111
x ^ y =  0001 0101
x << y = 1011 0000
x >> y =  0000 0010
~x      = 1110 1001
```

//-----

11.1.2 N: Prireditveni bitni operatorji

Z 8-bitnima številoma 22 in 3 ugotovimo rezultate prireditvenih aritmetičnih

`+=` `-=` `*=` `/=` `%=`

in prireditvenih bitnih operacij:

`&=` `|=` `^=` `<<=` `>>=`

Namen: Ločevanje med aritmetičnimi in bitnimi operacijami. Oblikovanje izpisa za bitne operacije.

Izpis:

```
x = 0x16 = 0001 0110 = 22
y = 0x03 = 0000 0011 = 03

Aritmetični operatorji:
x += y; x = 25
x -= y; x = 19
x *= y; x = 66
x /= y; x = 7
x %= y; x = 1

Bitni operatorji:
x &= y; x = 0000 0010
x |= y; x = 0001 0111
x ^= y; x = 0001 0101
x <<= y; x = 1011 0000
x >>= y; x = 0000 0010
```

//-----

11.1.3 N: Velikost tipov integer, short integer, long integer

Ugotovimo velikosti spremenljivk tipa *int*, *short int*, *long int* s pomočjo binarnih operatorjev. (Tip *int* nima povsod 32 bitov, tudi modifikacija *short int* nima povsod 16 bitov in modifikacija *long int* nima povsod 64 bitov. Npr., vgradni sistemi imajo tip *int* lahko tudi 16 biten).

Napišimo in uporabimo funkcije

```
int integerVelikost(),
int shortIntegerVelikost (),
int longIntegerVelikost (),
```

ki naj vrnejo velikosti spremenljivk tipov *int*, *short int* in *long int*.

Nasvet: ~ 0 rezultira v vrednosti vseh bitov 1. Z bitnim pomikom lahko ugotovimo, s koliko biti je spremenljivka zapisana.

Namen: Raba bitnih operacij. Dejansko ima isti tip spremenljivke na različnih platformah različno dolžino.

Izpis:

```
Integer spremenljivka ima na tem sistemu 32 bitov.
Short integer spremenljivka ima na tem sistemu 16 bitov.
Long integer spremenljivka ima na tem sistemu 32 bitov.
```

//-----

11.1.4 N: Iskanje bitnega vzorca

Naredimo funkcijo

```
int poiščiBitniVzorec (unsigned int Vir, unsigned int Vzorec,  
unsigned int nBitov);
```

Funkcija naj vrne mesto bita, kjer se v *Vir* začne *nBitov* iz *Vzorec*. Če iskanega zaporedja ni, naj funkcija vrne -1.

Primer:

```
index = poiščiBitniVzorec(0xe1f4, 0xF5, 3);
```

iščemo v 1110 0001 1111 0100

iščemo vzorec prvih 3 bitov Z LEVE, bit 1 je LSB, od 1111 0101, torej 101

vzorec iščemo začeni pri MSB. torej iščemo 101, začeni Z DESNE.

funkcija vrne številko 11: 101 se začne na 11em mestu: 1110 0001 1111 0100

Namen: Iskanje vzorca v bitnem nizu. Kompleksnejše delo z bitnimi operacijami.

Izpis:

```
Ko je stevilo 0XE1F4, vzorec 0xF5, stevilo relevantnih bitov 3, je index 11.
```

```
//-----
```

11.2 Rešitve

11.2.1 R: Bitni operatorji

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
void prikaziBin8(int x){  
    int rezultat, ostanek;  
    int digits[8] = {0,0,0,0,0,0,0,0};  
  
    int i=0;  
    do {  
        rezultat = x / 2;  
        ostanek = x % 2;  
        digits[i++]=ostanek;  
        x=rezultat;  
    } while (rezultat != 0);  
  
    for (i=7; i>=0; i--) {  
        if (i==3) putchar(' ');  
        if (digits[i]==1) putchar('1'); else putchar('0');  
    }  
}  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    unsigned char x = 0x16; // 8 bits  
    unsigned char y = 0x03;  
    unsigned char z;  
  
    z = x & y;  
    printf ("\nx = "); prikaziBin8(x); printf ("\n");  
    printf ("y = "); prikaziBin8(y); printf ("\n\n");
```



```
printf ("x & y = "); prikaziBin8(z); printf ("\n");
```

```
z = x | y;
```

```
printf ("x | y = "); prikaziBin8(z); printf ("\n");
```

```
z = x ^ y;
```

```
printf ("x ^ y = "); prikaziBin8(z); printf ("\n");
```

```
z = x << y;
```

```
printf ("x << y = "); prikaziBin8(z); printf ("\n");
```

```
z = x >> y;
```

```
printf ("x >> y = "); prikaziBin8(z); printf ("\n");
```

```
z = ~x;
```

```
printf ("~x = "); prikaziBin8(z); printf ("\n");
```

```
return 0;
```

```
}
```

```
//-----
```

11.2.2 R: Prireditveni bitni operatorji

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----  
  
void prikaziBin8(int x){  
    int rezultat, ostanek;  
    int digits[8] = {0,0,0,0,0,0,0,0};  
  
    int i=0;  
    do {  
        rezultat = x / 2;  
        ostanek = x % 2;  
        digits[i++]=ostanek;  
        x=rezultat;  
    } while (rezultat != 0);  
  
    for (i=7; i>=0; i--) {  
        if (i==3) putchar(' ');  
        if (digits[i]==1) putchar('1'); else putchar('0');  
    }  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    unsigned char x = 0x16; // 22  
    unsigned char y = 0x03; // 3  
  
    printf("\nx = 0x16 = 0001 0110 = 22 \n");  
    printf("y = 0x03 = 0000 0011 = 03 \n");  
  
    printf("\nAritmeticni operatorji:\n");
```

```

        printf("x += y; x = %d \n", x += y);    // 25
x = 0x16; printf("x -= y; x = %d \n", x -= y); // 19
x = 0x16; printf("x *= y; x = %d \n", x *= y); // 66
x = 0x16; printf("x /= y; x = %d \n", x /= y); // 7
x = 0x16; printf("x %%= y; x = %d \n", x %= y); // 1

printf("\nBitni operatorji:\n");
x = 0x16; printf("x &= y; x = "); prikaziBin8(x &= y); puts(""); // 0x02
x = 0x16; printf("x |= y; x = "); prikaziBin8(x |= y); puts(""); // 0x17
x = 0x16; printf("x ^= y; x = "); prikaziBin8(x ^= y); puts(""); // 0x15
x = 0x16; printf("x <<= y; x = "); prikaziBin8(x <<= y); puts(""); // 0xB0
x = 0x16; printf("x >>= y; x = "); prikaziBin8(x >>= y); puts(""); // 0x02

return 0;
}

//-----

```

11.2.3 R: Velikosti tipov int, short int, long int

```
//-----  
#include <stdio.h>  
#pragma hdrstop  
//-----
```

```
int integerVelikost (void)
```

```
{  
    unsigned int uiStevilo;  
    int iSteviloBitov = 0;  
    uiStevilo = ~0;
```

```
    while ( uiStevilo ) {  
        uiStevilo >>= 1;  
        iSteviloBitov++;
```

```
    }  
    return iSteviloBitov;  
}
```

```
int shortIntegerVelikost (void)
```

```
{  
    unsigned short int uiStevilo;  
    int iSteviloBitov = 0;  
    uiStevilo = ~0;
```

```
    while ( uiStevilo ) {  
        uiStevilo >>= 1;  
        iSteviloBitov++;
```

```
    }  
    return iSteviloBitov;  
}
```

```
int longIntegerVelikost (void)
```

```
{  
    unsigned long int uiStevilo;  
    int iSteviloBitov = 0;
```

```

uiStevilo = ~0;
while ( uiStevilo ) {
    uiStevilo >>= 1;
    iSteviloBitov++;
}
return iSteviloBitov;

```

```

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    int velikostInteger, velikostShortInteger, velikostLongInteger;

    velikostInteger = integerVelikost();
    printf ("\nInteger spremenljivka ima na tem sistemu %i bitov.\n",
    velikostInteger);

    velikostShortInteger = shortIntegerVelikost();
    printf ("Short integer spremenljivka ima na tem sistemu %i bitov.\n",
    velikostShortInteger);

    velikostLongInteger = longIntegerVelikost();
    printf ("Long integer spremenljivka ima na tem sistemu %i bitov.\n",
    velikostLongInteger);
    return 0;
}

//-----

```

11.2.4 R: Iskanje bitnega vzorca

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
unsigned int bitnaDolzinaJe(unsigned int uiStevilo)  
{  
    // z masko rastemo: 1, 11, 111, 111  
    // ko sta vhodno in maskirano stevilo enaka, imamo bitno dolzino stevila  
    // 00010111 maskirano z 1111 je 0111  
    // 00010111 maskirano z 11111 je 10111, torej: bitna dolzina je 5  
  
    unsigned int uiMaska, uiTemp, uiN;  
  
    uiMaska = 0;  
    for (uiN = 0; uiN <= 31; uiN++) {  
        uiMaska = uiMaska << 1 | 1;  
        uiTemp = uiStevilo & uiMaska;  
        if (uiTemp == uiStevilo)  
            return ++uiN;  
    }  
    return -1;  
}  
  
//-----  
int isciBitniVzorec (unsigned int uiSteviloPreiskovano, unsigned int  
uiSteviloVzorec, unsigned int uiStRelevBitovOdLSB)  
{  
    // v uiSteviloPreiskovano iscemo prvih uiStRelevBitovOdLSB z LSB strani  
    uiSteviloVzorec  
    // index je, kjer z MSB strani najdemo zacetek vzorca.  
    // ce ne najdemo, je return = -1
```

```

// uiSteviloVzorec maskiramo z uiStRelevBitovOdLSB
// iscemo enakost desno zamaknjena uiSteviloPreiskovano in maskiranega
uiSteviloVzorec

// test:
// uiSteviloPreiskovano = 0xE1F4 (1110 0001 1111 0100)
// uiSteviloVzorec = 0xF5          (1111 0101)
// uiStRelevBitovOdLSB =3

// return = 11

    unsigned int uiMaska, uiTemp, uiBitnaDolzina, uiSteviloVzorecMaskiran,
uiSteviloVzorecMaskiranZamaknjen, uiMaskaZamaknjena;
    unsigned int n;

    if (uiStRelevBitovOdLSB > 32) return -1;

    uiBitnaDolzina = bitnaDolzinaJe(uiSteviloPreiskovano);
    if (uiBitnaDolzina == -1) return -1;

    uiMaska = 0;          // kreiranje maske za pobrati vzorec, ki ga iscemo
    for (n = 0; n < uiStRelevBitovOdLSB; n++)
        uiMaska = uiMaska << 1 | 1;

    // pobran vzorec, ki ga iscemo
    uiSteviloVzorecMaskiran = uiSteviloVzorec & uiMaska;

    // vzorec zapeljemo do MSB
    uiSteviloVzorecMaskiranZamaknjen = uiSteviloVzorecMaskiran <<
(uiBitnaDolzina - uiStRelevBitovOdLSB);
    // masko zapeljemo do MSB
    uiMaskaZamaknjena = uiMaska << (uiBitnaDolzina - uiStRelevBitovOdLSB);

    // imamo Masko,
    // imamo N mest iz Vzorca,
    // pomikamo Zacetno stevilo v levo, torej proti MSB

```

```

// ko je MASKIRANO pomikano zacetno stevilo enako N mestom iz Vzorca,
// imamo Index za v Return(Index).
uiTemp = 0;
for (n=0; n < uiBitnaDolzina; n++) {
    uiSteviloPreiskovano = uiSteviloPreiskovano << 1;
    uiTemp = uiSteviloPreiskovano & uiMaskaZamaknjena;
    if (uiTemp == uiSteviloVzorecMaskiranZamaknjen)
        return ++n;
}
return -1;
}

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    unsigned int uiSteviloPreiskovano, uiSteviloVzorec, uiStRelevBitovOdLSB;
    int isciBitniVzorec (unsigned int uiSteviloPreiskovano, unsigned int
uiSteviloVzorec, unsigned int uiStRelevBitovOdLSB);

    uiSteviloPreiskovano = 0XE1F4;
    uiSteviloVzorec = 0XF5;
    uiStRelevBitovOdLSB = 3;

    printf("Ko je stevilo 0X%X, vzorec 0x%X, stevilo relevantnih bitov %i, je
index %i.\n",
        uiSteviloPreiskovano, uiSteviloVzorec, uiStRelevBitovOdLSB,
        isciBitniVzorec (uiSteviloPreiskovano, uiSteviloVzorec,
uiStRelevBitovOdLSB));

    return 0;
}

//-----

```


12 Datotečne funkcije

Poudarki:

- Odpiranje, pisanje in zapiranje datoteke, funkcije *fopen()*, *fputs()*, *fclose()* iz *stdio* knjižnice.
- Kopiranje datoteke po bytih, *getc()*, *putc()*, in po blokih, *fread()*, *fwrite()* iz knjižnice *stdio*.
- Spreminjanje podatkov v datoteki.
- Ugotovitev velikosti datoteke iz prebranega števila znakov.
- Združevanje podatkov iz različnih datotek.
- Oblikovanje izpisov podatkov iz datotek.

12.1 Naloge

12.1.1 N: Pisanje v datoteko

Napišimo program, ki naj najprej naredi datoteko *besedilo.txt*. V to datoteko naj vpiše:

To je prvi stavek.

To je DRUGI stavek.

Datoteko naj zapre, spet odpre in vpiše naj:

To je tretji stavek.

Datoteko naj zapre.

Z Notepad.exe preverimo vsebino *besedilo.txt*.

Namen: odpiranje, pisanje, zapiranje datoteke. Funkcije *fopen()*, *fputs()*, *fclose()*.

Izpis:

besedilo.txt:

```
To je prvi stavek.  
To je DRUGI stavek.  
To je tretji stavek.
```

//-----

12.1.2 N: Vpis in izpis

Naredimo program, s katerim bomo vpisovali imena in višine oseb v datoteko *osebe.dat*. To datoteko naj s tem programom tudi beremo.

Namen: Naredimo strukturo niza imena in *int* starosti. Ustrezen podatek vpisujemo. Datoteko beremo v enako spremenljivko in jo izpisujemo. Pisanje in branje s konzolo in z datoteko.

Izpis:

```
Upis U ali Izpis I ali konec K:
U
Upisite ime:
Slavica
Upisite visino:
179
Upis U ali Izpis I ali konec K:
Upis U ali Izpis I ali konec K:
I
Anton          188
Stefan         182
Melanija       166
Slavica        179
Upis U ali Izpis I ali konec K:
K
Konec.
```

//-----

12.1.3 N: Kopiranje datoteke po bytih

Naredimo program za kopiranje prve v drugo datoteko po bytih. Program kličemo z: *ime_programa ime_datoteke ime_kopije_datoteke*
Sporočila v primeru ugotovljenih napak naj bodo:

Uporaba: *ImePrograma <IzvornaDatoteka> <CiljnaDatoteka>\n*

Izvirne datoteke ne morem odpreti.

Ciljne datoteke ne morem odpreti.

Napaka pri branju.

Napaka pri pisanju.

Namen: Branje parametrov programa iz ukazne vrstice. Branje in pisanje datoteke po bytih, detekcija možnih napak pri branju in pisanju datotek (ugotavljanja zastavic napak in njihovo resetiranje).

Izpis:

```
D:\>Project1 video.ts kopija.ts
Uhodni parametri(0):, D:\Project1.exe
Uhodni parametri(1):, video.ts
Uhodni parametri(2):, kopija.ts
```

Kopija je identična originalu. V danem primeru je datoteka kratek video; predvaja se enako kot original.

//-----

12.1.4 N: Kopiranje datoteke po blokih

Naredimo program za kopiranje prve v drugo datoteko po 1024 bytnih blokih. Program kličemo z:

ime_programa ime_datoteke ime_kopije_datoteke

Sporočila v primeru ugotovljenih napak naj bodo:

Uporaba: *ImePrograma <IzvornaDatoteka> <CiljnaDatoteka>\n*

Izvirne datoteke ne morem odpreti.

Ciljne datoteke ne morem odpreti.

Napaka pri branju.

Napaka pri pisanju.

Namen: Branje parametrov programa iz ukazne vrstice. Branje in pisanje datoteke po blokih, detekcija možnih napak pri branju in pisanju datotek (ugotavljanja zastavic napak in njihovo resetiranje).

Kopiranje po blokih je na vgradnih sistemih hitrejše od kopiranja po bytih. Na delovni postaji sta časa primerljiva; večkrat zapored uporabljane datoteke so v delovnem spominu in operacijski sistem sam izvaja sinhronizacijo med delovnim spominom in trdim diskom.

Izpis:

```
D:\>Project1 video.ts kopija.ts
Uhodni parametri(0):, D:\Project1.exe
Uhodni parametri(1):, video.ts
Uhodni parametri(2):, kopija.ts
```

Kopija je identična originalu. V danem primeru je datoteka kratek video; predvaja se enako kot original.

//-----

12.1.5 N: Sprememba v velike črke

Napišimo program, ki bo skopiral prvo datoteko v drugo in pri tem vse male črke spremenil v velike črke.

Namen: Odpiranje, branje, pisanje, zapiranje datotek. Spremembo velikosti črk naredimo sami ali z obstoječo funkcijo.

Izpis:

```
Ime datoteke, ki jo kopirate: datotekaMesano.txt
Ime datoteke, v katero kopirate: rezultat.txt
Datoteka je skopirana.
```

datotekaMesano:

```
abcdefghijklmnopqrstuvwxyz 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwxyz 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwxyz 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwxyz 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 abcdefghijklmnopqrstuvwxyz
```

rezultat.txt:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ

ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ

ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ

ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

//-----

12.1.6 N: Velikost datoteke

Ugotovimo velikost datoteke z branjem po en znak do *EOF* (End Of File – zadnji znak v datoteki, običajno je *EOF* -1, sistem ta znak sam napiše ob zaprtju datoteke) in izpišimo število prebranih znakov.

Vpišite ime datoteke, za katero ugotavljamo velikost:

Dolžina datoteke je n bytov.

ali, če težava: Napaka pri odpiranju datoteke.

Namen: Odpiranje, zapiranje datoteke, branje po en byte v zanki in sprotno preverjanje prebranega znaka.

Izpis:

```
Upisite ime datoteke, za katero ugotavljamo velikost: video.ts
Dolzina datoteke je 4014364 bytov.
```

//-----

12.1.7 N: Izpis meritev

Datoteka "*meritev.txt*" predstavlja log meritve. V vsaki vrstici je zapisan najprej datum meritve v obliki dd.mm.yyyy. Sledi presledek, sledi realno število.

Napišite funkcijo, ki ugotovi, kdaj je bila izmerjena vrednost najmanjša. Funkcija naj za parametre dobi ime vhodne datoteke in tri kazalce na cela števila, prek katerih vrne datum najmanjše meritve.

Deklaracija funkcije:

```
int DatumMinimума(char *imeDat, int *dan, int *mesec, int *leto);
```

Funkcija naj vrne uspešnost izvedbe:

0: v redu,

1: datoteke ni možno odpreti,

2: datoteka je prazna in najmanjše vrednosti ni možno najti.

S programom *main()* preverite delovanje funkcije.

Namen: Branje številske datoteke. Ustrezno formatiranje branja.

Izpis:

```
Najmanjsa izmerjena vrednost je na dan/mesec/leto: 24.12.2014.
```

meritev.txt:

1.12.2014	20.55
2.12.2014	20.65
3.12.2014	20.75
4.12.2014	20.85
5.12.2014	20.95
6.12.2014	21.05
7.12.2014	21.15
8.12.2014	21.25
9.12.2014	21.35
10.12.2014	21.45
11.12.2014	21.35
12.12.2014	21.25
13.12.2014	21.15
14.12.2014	21.05
15.12.2014	20.95
16.12.2014	20.85
17.12.2014	20.75
18.12.2014	20.65
19.12.2014	20.55
20.12.2014	20.45
21.12.2014	20.35
22.12.2014	20.25
23.12.2014	20.15
24.12.2014	20.05

//-----

12.1.8 N: Izpis po n vrstic

Naredimo program, ki izpiše po 20 vrstic iz datoteke naenkrat.

Namen: Oblikovanje izpisa podatkov v datoteki.

Izpis:

```
Upisite ime datoteke: dat.txt
dat.txt, vrsta 1 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 2 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 3 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 4 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 5 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 6 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 7 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 8 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 9 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 10 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 11 od 75: A
dat.txt, vrsta 12 od 75: 0123456789
dat.txt, vrsta 13 od 75: B
dat.txt, vrsta 14 od 75: ~!@#%&*( )_+!
dat.txt, vrsta 15 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 16 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 17 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 18 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 19 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 20 od 75: abcdefghijklmnopqrstuvwxyz
Upisite 999 za konec, drugo poljubno stevilo za nadaljevanje izpisa:
1
dat.txt, vrsta 21 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 22 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 23 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 24 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 25 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 26 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 27 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 28 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 29 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 30 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 31 od 75: A
dat.txt, vrsta 32 od 75: 0123456789
dat.txt, vrsta 33 od 75: B
dat.txt, vrsta 34 od 75: ~!@#%&*( )_+!
dat.txt, vrsta 35 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 36 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 37 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 38 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 39 od 75: abcdefghijklmnopqrstuvwxyz
dat.txt, vrsta 40 od 75: abcdefghijklmnopqrstuvwxyz
Upisite 999 za konec, drugo poljubno stevilo za nadaljevanje izpisa:
999
Konec.
```


12.2 Rešitve

12.2.1 R: Pisanje v datoteko

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    FILE *pFile;  
    pFile = fopen ("besedilo.txt", "w");  
    if (pFile != NULL)  
    {  
        fputs("To je prvi stavek.\n", pFile);  
        fputs("To je DRUGI stavek.\n", pFile);  
        fclose (pFile);  
    }  
    pFile = fopen ("besedilo.txt", "a"); // a: append  
    if (pFile != NULL)  
    {  
        fputs("To je tretji stavek.\n", pFile);  
        fclose (pFile);  
    }  
  
    return 0;  
}  
//-----
```

12.2.2 R: Vpis in izpis

```
//-----  
#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
struct oseba {  
    char ime[30];  
    int visina;  
};  
  
int vpis(void);  
int izpis(void);  
  
//-----  
int vpis(){  
    struct oseba os;  
    int a;  
    FILE *pDatoteka;  
    char ime[30];  
  
    puts("Vpisite ime:");  
    gets( os.ime );  
    puts("Vpisite visino: \n");  
    scanf("%i", &os.visina);  
  
    if ((pDatoteka=fopen("osebe.dat","a")) == NULL) { // a: append  
        printf("Napaka, fopen().\n");  
        return 1;  
    }  
  
    fwrite(&os,sizeof(struct oseba),1,pDatoteka);
```

```

fclose(pDatoteka);
return 0;
}

//-----

int izpis(){
FILE *pDatoteka;
struct oseba r;

if ((pDatoteka=fopen("osebe.dat","r")) == NULL) {
printf("Napaka, fopen().\n");
return 1;
}

while(!feof(pDatoteka)) {
if (!fread(&r, sizeof(struct oseba),1,pDatoteka) )
break;
printf( "%-12s %d ", r.ime, r.visina);
printf("\n");
}

fclose(pDatoteka);
return 0;
}

//-----
#pragma argsused
int main(int argc, char* argv[])
{
char clzbira[10];
char ime[40];
while (true) {
puts ("Vpis V ali Izpis I ali konec K: \n");
gets(clzbira);
if (clzbira[0] == 'V') {

```

```
    vpiš(); }
else if (clzbira[0] == 'I')
    izpiš();
else if (clzbira[0] == 'K')
    break;
else ;
}

puts ("Konec.");

return 0;
}
//-----
```

12.2.3 R: Kopiranje datoteke po bytih

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
FILE *pBerem, *pPisem;  
char ch;  
int i;  
  
puts("");  
for (i=0; i < argc; i++)  
    printf ("Vhodni parametri(%i):, %s\n", i, argv[i]);  
  
if(argc!=3) {  
    printf("Uporaba: ime_programa <izvorna_datoteka> <ciljna_datoteka>\n");  
    return -1;  
}  
  
if((pBerem=fopen(argv[1], "rb")) == NULL) { // read, binary  
    printf("Napaka, izvorna datoteka, fopen().\n");  
    return -2;  
}  
if((pPisem=fopen(argv[2], "wb")) == NULL) {  
    printf("Napaka, ciljna datoteka, fopen().\n");  
    return -3;  
}  
  
while(!feof(pBerem)) {  
    ch = getc(pBerem);  
    if(ferror(pBerem)) { // Flag Error
```

```
    printf("Napaka pri branju");
    clearerr(pBerem);
    return 1;
} else {
    putc(ch, pPisem);
    if(ferror(pPisem)) {
        printf("Napaka pri pisanju");
        clearerr(pPisem);
        return 2;
    }
}

fclose(pBerem);
fclose(pPisem);

return 0;
}
//-----
```


12.2.4 R: Kopiranje datoteke po blokih

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    FILE *pBerem, *pPisem;  
    char buffer[1024]; // 2**10  
    size_t prebrano; // unsigned integral type, unsigned int  
    int i;  
  
    puts("");  
    for (i=0; i < argc; i++)  
        printf ("Vhodni parametri(%i):, %s\n", i, argv[i]);  
  
    if(argc!=3) {  
        printf("Uporaba: ImePrograma KopiranaDatoteka KopijaDatoteke\n");  
        return -1;  
    }  
  
    if((pBerem=fopen(argv[1], "rb")) == NULL) { // read binary  
        printf("Napaka, prva datoteka, fopen().\n");  
        return -2;  
    }  
    if((pPisem=fopen(argv[2], "wb")) == NULL) { // write binary  
        printf("Napaka, druga datoteka, fopen().\n");  
        return -3;  
    }  
  
    while(!feof(pBerem)) {  
        prebrano = fread( buffer, 1, 1024, pBerem);
```

```
if(ferror(pBerem)) {
    printf("Napaka pri branju");
    clearerr(pBerem);
    return 1;
} else {
    if (prebrano>0)
        fwrite(buffer, 1, prebrano, pPisem);
    if(ferror(pPisem)) {
        printf("Napaka pri pisanju");
        clearerr(pPisem);
        return 2;
    }
}
fclose(pBerem);
fclose(pPisem);

return 0;
}
//-----
```

12.2.5 R: Sprememba v velike črke

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#include <ctype.h>  
#pragma hdrstop  
//-----  
#pragma argsused  
int main(int argc, char* argv[]) {  
    char imeVhodne[64], imelzhodne[64];  
    FILE *pBerem, *pPisem;  
    int c;  
  
    printf ("\nIme datoteke, ki jo kopirate: ");  
    scanf ("%63s", imeVhodne);  
    printf ("Ime datoteke, v katero kopirate: ");  
    scanf ("%63s", imelzhodne);  
  
    if ( (pBerem = (FILE *) fopen (imeVhodne, "r")) == NULL ) {  
        fprintf (stderr, "Ne morem odpreti %s za branje.\n", imeVhodne);  
        return 1; }  
    else if ( (pPisem = fopen (imelzhodne, "w")) == NULL ) {  
        fprintf (stderr, "Ne morem odpreti %s za pisanje.\n", imelzhodne);  
        return 2; }  
  
    while ( (c = getc (pBerem)) != EOF )  
        putc (toupper(c), pPisem);  
  
    fclose (pBerem);  
    fclose (pPisem);  
    printf ("Datoteka je skopirana.\n");  
    return 0;  
}  
//-----
```

12.2.6 R: Velikost datoteke

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    FILE *pDatoteka;  
    unsigned int n = 0;  
    char ime[255];  
    printf( "\nVpisite ime datoteke, za katero ugotavljamo velikost: " );  
    gets(ime);  
    pDatoteka = fopen (ime,"rb");  
    if (pDatoteka==NULL) perror ("Napaka pri odpiranju datoteke.");  
    else  
    {  
        while (!feof(pDatoteka)) {  
            fgetc (pDatoteka);  
            n++;  
        }  
        fclose (pDatoteka);  
        printf ("Dolzina datoteke je %u bytov.\n",--n); // u - unsigned  
    }  
    return 0;  
}  
  
//-----
```

12.2.7 R: Izpis meritev

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#define MAX_DOLZINA 100  
#pragma hdrstop  
//-----  
  
int DatumMinimuma(char *plmeDatoteke, int *pDan, int *pMesec, int *pLeto)  
{  
    FILE *pDatoteka;  
    int stevec = 0;  
    int d, m, l;  
    double podatek, podatek_min;  
  
    pDatoteka = fopen(plmeDatoteke, "r");  
    if (pDatoteka == NULL) return 1;  
    while (true) {  
        if (fscanf(pDatoteka, "%d.%d.%d %lf", &d, &m, &l, &podatek) != 4) break;  
        if (stevec == 0 || podatek < podatek_min) {  
            podatek_min = podatek;  
            *pDan = d; *pMesec = m; *pLeto = l;  
        }  
        stevec++;  
    }  
    if (stevec == 0) return 2;  
    fclose(pDatoteka);  
    return 0;  
}  
  
//-----  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    int dan, mesec, leto;
```

```
int ret = DatumMinimума("meritev.txt", &dan, &mesec, &leto);
if (ret == 1) printf("Ne morem odpreti datoteke!\n");
else if (ret == 2) printf("Datoteka je prazna!\n");
else printf("\nNajmanjsa izmerjena vrednost je na
dan/mesec/leto: %d.%d.%d.\n", dan, mesec, leto);

return 0;
}

//-----
```

12.2.8 R: Izpis po *n* vrstic

```
//-----  
#include <stdio.h>  
#include <stdbool.h>  
#pragma hdrstop  
//-----  
  
#pragma argsused  
int main(int argc, char* argv[])  
{  
    #define DIM 80  
    #define IZPIS_STOP 20  
  
    char imeDatoteke[64];  
    FILE *pDatoteka;  
    int iZnak, iIndexZnakaVVrsti, iIndexPrikazaneVrste, ilnit;  
    int iVnosUporabnika;  
    char cVrsta[DIM];  
    for (ilnit=0; ilnit<DIM; ilnit++) {  
        cVrsta[ilnit]=0;  
    }  
  
    printf ("\nVpisite ime datoteke: ");  
    scanf ("%63s", imeDatoteke);  
  
    if ( (pDatoteka = fopen (imeDatoteke, "r")) == NULL ) {  
        fprintf (stderr, "Datoteka %s ni odprta za branje, napaka.\n", imeDatoteke);  
        return 1; }  
  
    iIndexZnakaVVrsti = 0; iIndexPrikazaneVrste = 0;  
    do {  
        iZnak = getc(pDatoteka);  
        if (iZnak != '\n' && iZnak != EOF)  
            cVrsta[iIndexZnakaVVrsti++] = iZnak;  
        else {
```

```

cVrsta[iIndexZnakaVVrsti++] = '\n'; cVrsta[iIndexZnakaVVrsti++] = NULL;
printf ("%s", cVrsta);
iIndexZnakaVVrsti=0;
iIndexPrikazaneVrste++;
}

if (iIndexPrikazaneVrste == IZPIS_STOP) {
    printf ("Vpisite 999 za konec, drugo poljubno stevilo za nadaljevanje izpisa:
\n");
    scanf ("%i", &iVnosUporabnika);
    iIndexPrikazaneVrste = 0;
    if (iVnosUporabnika == 999) break; }
}
while (iZnak != EOF);

fclose (pDatoteka);
printf ("Konec.\n");

return 0;
}
//-----

```


13 Literatura

- [1] I. Sommerville, Software Engineering, 9. izdaja, Addison Wesley 2011, ISBN 0137035152
- [2] C. Jones, The Technical and Social History of Software Engineering, Addison Wesley 2014, ISBN 0321903420
- [3] W. Suryn, Software Quality Engineering: A Practitioner's Approach, Wiley & Sons 2014, ISBN 9781118592496
- [4] M. Barr, A. Massa, Programming Embedded Systems, O'Reilly 2006, ISBN 0596009836
- [5] K. H. John, M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems, 2. izdaja, Springer 2000. ISBN 9783642120145
- [6] B. W. Kernighan, D. M. Ritchie, The C Programming Language, ANSI C, 2. izdaja, Prentice Hall 1988, ISBN 0131103628
- [7] Prevod B.W.Kernighan, D.M.Ritchie, Programski jezik C, prevajalec Leon Mlakar, 3. izdaja, FER v Ljubljani, 1991
- [8] S. Prata, C Primer Plus, 6. izdaja, Addison Wesley 2014, ISBN 0321928423
- [9] S. G. Kochan, Programming in C, 4. izdaja, Addison Wesley 2014, ISBN 0321776410
- [10] <http://www.cplusplus.com/reference/clibrary/> , dostop dec. 2014
- [11] T. Dobravec, abC, FRI v Ljubljani, 2. izdaja, 2010, ISBN 9789616209755
- [12] <http://www.c4learn.com/c-programs/> , dostop dec. 2014