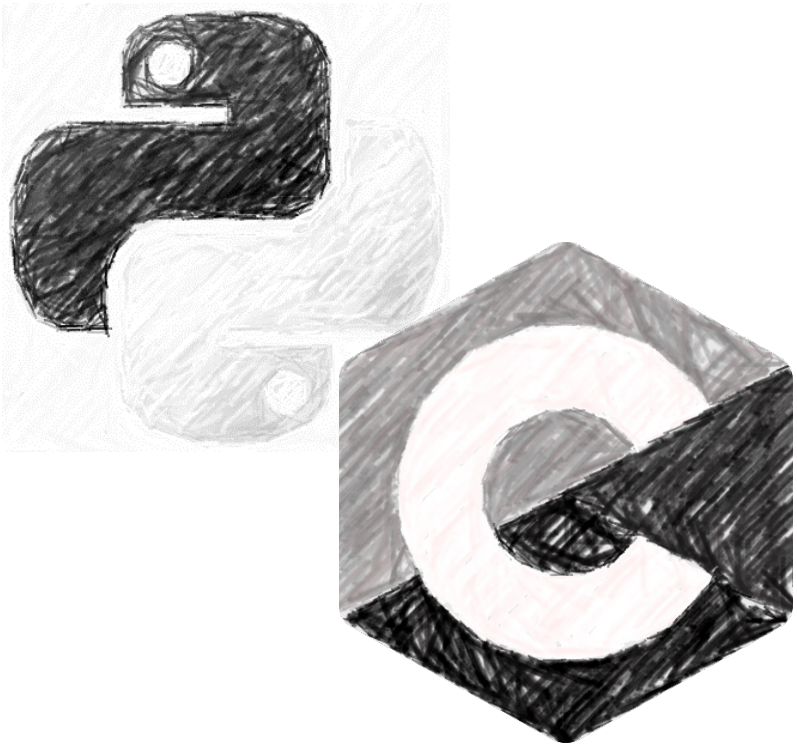
Izvleček navodil za programiranje v programskem jeziku Python in C



Inštitut za avtomatiko

Fakulteta za elektrotehniko, računalništvo in informatiko

Univerza v Mariboru

Andrej Sarjaš

Maj 2023

Podatki o delu:

Naslov:	*Izvleček navodil za programiranje v programskem jeziku Python in C*
Avtor:	*doc. dr. Andrej Sarjaš, univ. dipl. inž. el.*
Strokovni recenzent:	*prof. dr. Dušan Gleich, univ. dipl. inž. el.*
Lektor:	*Tanja Bigec, prof. slovenščine*
Založnik:	*Fakulteta za elektrotehniko, računalništvo in informatiko Maribor*
Vrsta publikacije:	*Gradivo za vaje*
Tisk:	*Elektronski vir (pdf)*
Način dostopa:	*https://estudij.um.si/*
Leto izdaje:	*Maribor, maj 2023*

Gradivo je namenjeno računalniškim vajam pri predmetih Računalništvo, Regulacije 1 in 2, Regulacijska tehnika 2, Napredni regulacijski sistemi in izdelava zaključnega dela na Inštitutu za avtomatiko, FERI, UM. Prav tako gradivo služi kot pomoč pri študiji in učenju vsebin, pri katerih je potrebno interdisciplinarno znanje z več področij. Prav znanje programiranja je simbioza, ki učinkovito združuje več tehničnih in znanstvenih disciplin v danem času.

Kazalo:

Tabele

Python

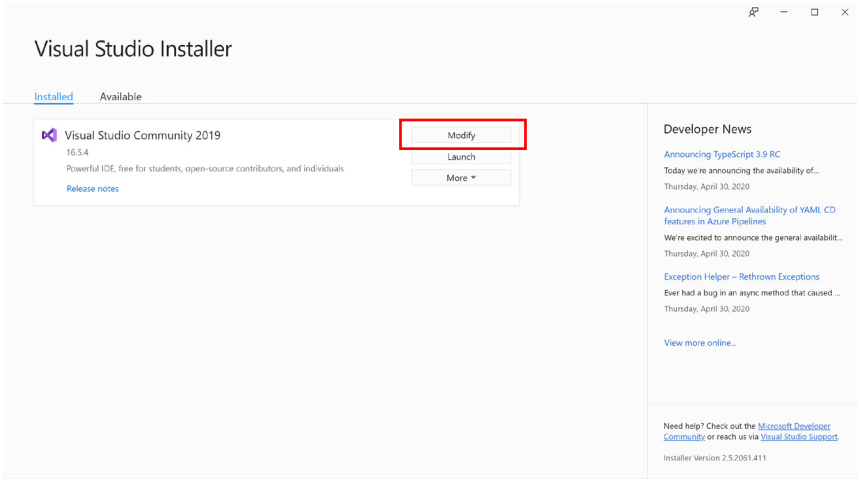
1. Uvod v Python

Python je interpreterski programski jezik, ki ga je ustvaril Guido van Rossum leta 1990 [1]. Jezik je dobil ime po priljubljeni angleški televizijski nanizanki Leteči cirkus Montyja Pythona[1]. Python ima popolnoma dinamične podatkovne tipe, samodejno upravlja s pomnilnikom in podpira funkcionalno, imperativno oziroma proceduralno, strukturirano in objektno orientirano računalniško programsko paradigmo. Zaradi dinamičnih podatkovnih tipov je podoben jezikom Perl, Ruby, Scheme, Smalltalk in Tcl. Razvili so ga kot odprtokodni projekt, ki ga je upravljala neprofitna organizacija Python Software Foundation. Python se v glavnem uporablja za računalniško analitiko in razvijanje internetnih aplikacij.

Python uporablja različna orodja za različne tipe razvijanja aplikacij. Za grafični uporabniški vmesnik oziroma GUI[2] ima več različnih orodij, kot so: Kivy, PyQt, PySide, Pygame, PyForms, TkInter. Python ima tudi orodja za razvijanje internetnih aplikacij. Med najpogosteje uporabljena sodijo: Django, ki ima svojo posebno knjižnico in deluje večinoma v povezavi s podatkovnimi bazami, Bottle, Flask, Tornado (Orodje), web2py in Jade. Večina teh orodji je že nameščena, če uporabnik uporablja Integrirano Razvijalsko Okolje oziroma IDE. Python pa ima tudi orodja za analitiko in razvijalce programske opreme. Med najpogosteje uporabljena orodja za analitiko spadajo: SciPy, Pandas in IPython. Za razvijanje programske opreme se uporabljajo predvsem naslednja orodja: Buildbot, Trac in Roundup. Uporabljajo se tudi orodja za skrbništvo sistema, kot so na primer: Ansible, Salt in OpenStack.

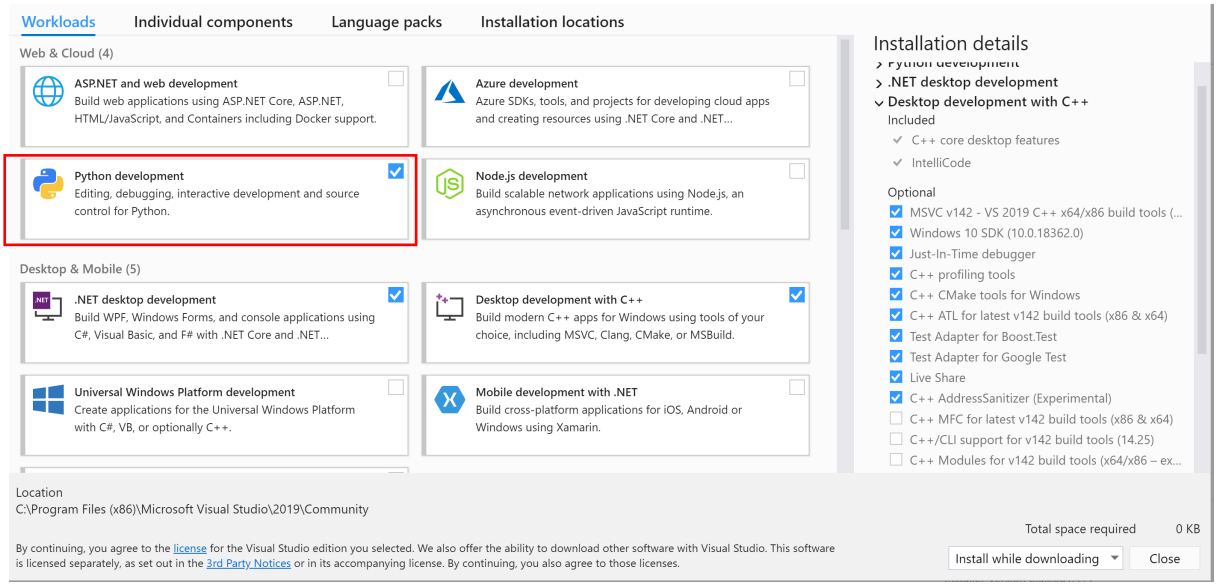
2. Nastavitev Visual Studio 2022 za uporabo programskega jezika Python

Če imamo že prednaloženo programsko okolje Visual Studion 2022 (VS2022), v **Visual Studio Installer** preverimo ali imamo dodan paket za programski jezik Python. **Visual Studio Installer** najdemo med naloženimi programi, kjer je naložen tudi VS2022.



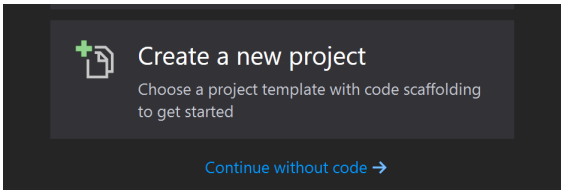
[1] Monthy Python's Flying Circus.
[2] GUI-Graphical user interface.

Pod tipko ***Modify*** preverimo nameščene pakete, če še nimamo naloženega Python, ga lahko dodatno namestimo.

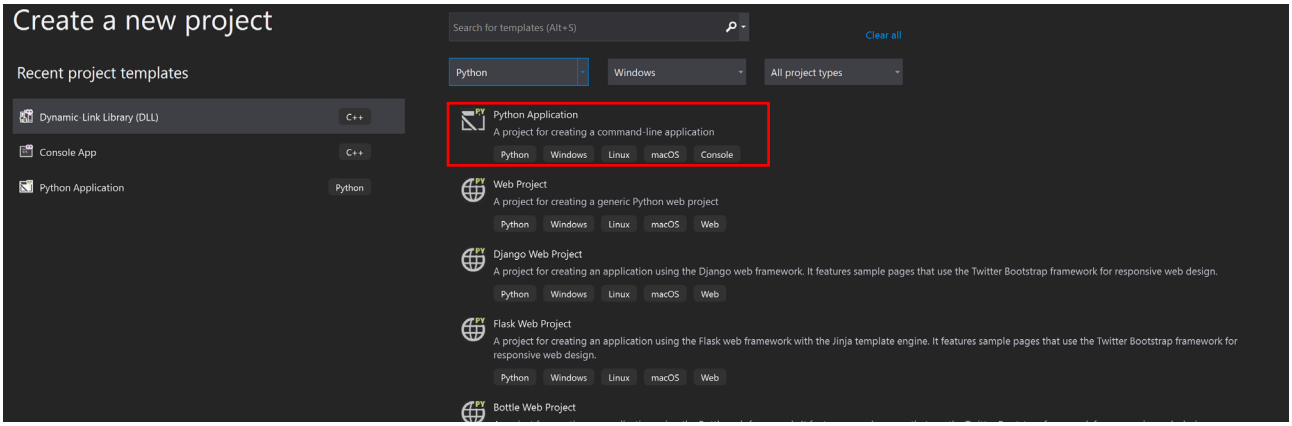


Kreiranje novega projekta v programskem okolju Python

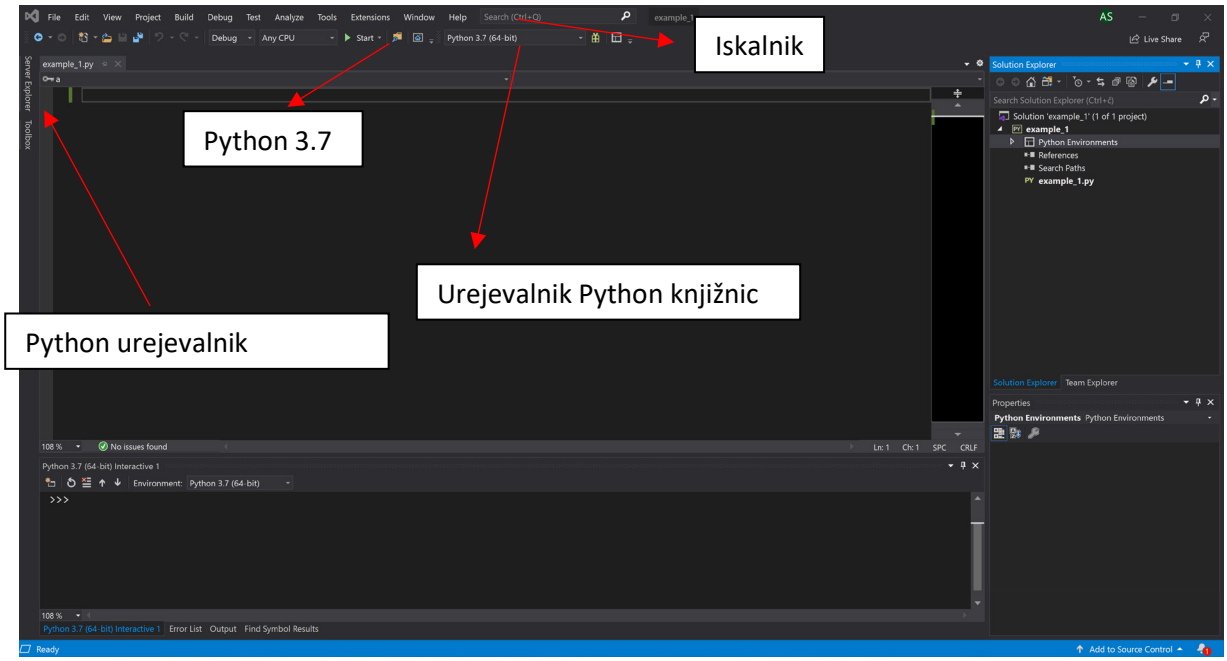
Po namestitvi Python razvijalskega paketa lahko pričnemo s kreiranjem novega projekta. Izberemo **Create a new project**:



Kjer izberemo **Python Application**:

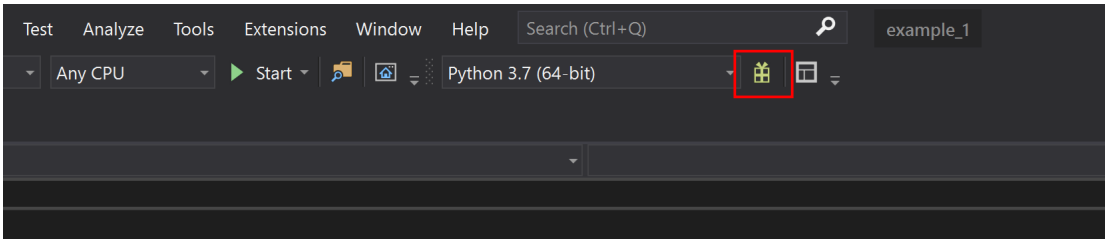


Odprti projekt:



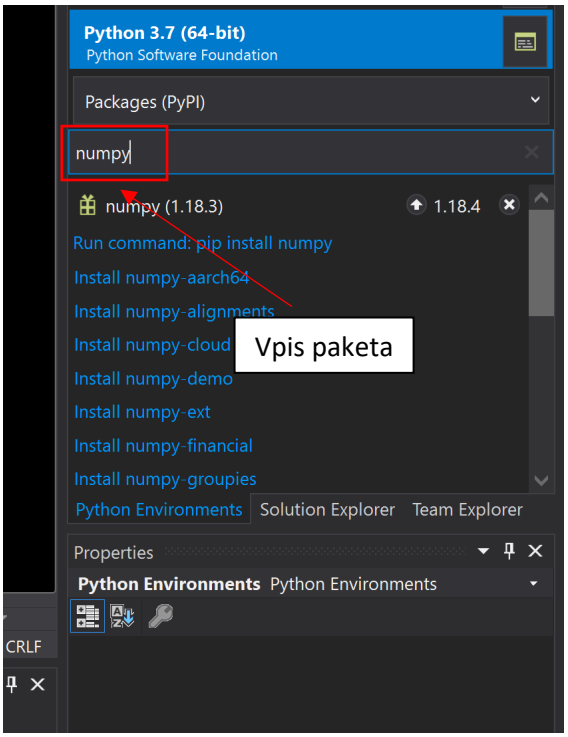
Po odprtju projekta lahko izberemo, katero okolje Pythona želimo uporabljati. Izberemo na način, da sledimo **View** > **Other Windows** > **Python Environments** ali enostavno v iskalnik vpišemo **Python Environments**. Če želimo uporabljati privzeto okolje, tega ne spreminjamo.

Interpreterski jezik Python omogoča veliko različnih knjižnic za matematična operacije, izris grafov, rokovanje z različnimi vtičniki ter periferijo na računalniškem sistemu, kot so serijski vmesniki, ethernet, usb vmesnik, bluetooth itd. Prav tako ima bogato shrambo za grafične vmesnike, zajemanje slik ter orodja umetne inteligence. Pakete naložimo s klikom na  ali v iskalnik vpišemo **Manage** Python **packages** .



Izberemo knjižnico, ki jo želimo namestiti. Za začetek bomo namestili knjižnice: numpy, matplotlib, tkinter, time itd.

Okno za namestitev paketa.



Knjižnico v skripto vključimo z ukazom 'import'. Pri pisanju skripte moramo biti pozorni na zamik vrstic. Zamik vrstic določa, ali se ukaz izvaja pod določeno funkcijo, zanko ali pogojem.

3. Spremenljivke, nizi, operatorji in polja

Spremenljivke, nizi in podatkovni tipi

Primer: Spremenljivke in nizi (string).

```
x = 7.2    #stevilo
y = 2
z = "Miha" #string
print(x)   #izpis
print(y)
print(z)
print(z[1]) #izpis drugega elementa iz niza >>i


x, y, z = "janko", "metka", "čarovnica"

print(x," ",y," ",z) #izpis
```

Pogoste operacije s stringi

```
str = "Hello, World!"
print(len(a))         #Dolzona niza
print(str.split(",")) #Razdelimo niz kjer je vejica, rezultat>>['Hello', ' World!']
print(str.replace("H", "J"))#Zamenjava H za J >> ‘Hello, World’
```

Podatkovni tipi:

tekstovni:	`str`
numerični:	`int`, `float`, `complex`
sekvenčni:	`list`, `tuple`, `range`
mapirni:	`dict`
množice:	`set`, `frozenset`
bulovi:	`bool`
binearni:	`bytes`, `bytearray`, `memoryview`

Funkcij type() vrne podatkovni tip spremenljivke.

```
num = 7
print(type(num)) #izpiše int
```

Tabela 1: Podatkovni tipi

Primer	Podatkovni tip
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset

x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Primer: Sprememba podatkovnega tipa (Casting) v celo število.

```
x = int(1)   # x bo 1
y = int(2.8) # y bo  2
z = int("3") # z bo 3
```

V decimalno število:

```
x = float(1)     # x bo 1.0
y = float(2.8)   # y bo 2.8
z = float("3")   # z bo 3.0
w = float("4.2") # w bo 4.2
```

V niz/string število:

```
x = str("s1") # x bo 's1'
y = str(2)    # y bo '2'
z = str(3.0)  # z bo '3.0'
```

Operatorji

Operatorje uporabljamo za rokovanje s spremenljivkami. Tako poznamo naslednjo vrsto operatorjev [2]-[4]:

- **Aritmetični operatorji**
- **Določitveni operatorji**
- **Primerjalni operatorji**
- **Logični operatorji**
- **Enotski operatorji**
- **Pripadnosti operatorji**
- **Bitni operatorji**

Aritmetični operatorji

Tabela 2: Aritmetični operatorji

Operator	Ime	Primer
+	seštevanje	x + y
-	odštevanje	x - y
*	množenje	x * y
/	deljenje	x / y
%	ostanek	x % y
**	eksponent	x ** y
//	deljenje z zaokrožanjem navzdol	x // y

Določitveni operatorji

Tabela 3: Določitveni operatorji

Operator	Ime	Primer
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3

^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Primerjalni operatorjiTabela 4

Tabela 5: Primerjalni Operatorji

Operator	Ime	Primer
==	enako	x == y
!=	neenako	x != y
>	večje	x > y
<	manjše	x < y
>=	večje ali enako	x >= y
<=	manjše ali enako	x <= y

Logični operatorji

Tabela 6: Logični Operatorji

Operator	Ime	Primer
and	Vrne **True**, če oba ob pogoja držita, sicer **False.**	x < 5 and x < 10
or	Vrne **True**, če vsaj en od pogojev drži, sicer **False.**	x < 5 or x < 4
not	Vrne **False**, če sta pogoja **True.**	not(x < 5 and x < 10)

Enotski operatorji

Tabela 7: Enotski operatorji

Operator	Ime	Primer
is	Vrne **True**, če sta obe spremenljivki enak objekt ali tip, sicer **False.**	x is y
is not	Vrne **True**, če obe spremenljivki nista enak objekt ali tip, sicer **False.**	x is not y

Pripadnosti operatorjiTabela 8

Tabela 9: Pripadnostni operatorji

Operator	Ime	Primer
in	Vrne **True**, če je niz ali sekvenca y v x.	x in y
not in	Vrne **True**, če je niz ali sekvenca y ni v x.	x not in y

Bitni operatorji

Tabela 10: Bitni operatorji

Operator	Ime	Opis
&	AND	IN-operator
|	OR	ALI-operator
^	XOR	ekskluzivni ALI
~	NOT	Invertiranje bitov
<<	Pomik v levo, dodaj ničle.	Pomik v levo, kjer dodajmo ničle, skrajno levi biti odpadejo.

>>	Pomik v desno	Pomik v desno, kjer pomikamo najvišje ležeče leve bite v desno, skrajno desni pa odpadejo.

Polja

V Pythonu poznamo štiri vrste polj, ki jih delimo na:

• **List (seznam)** je polje, ki je urejeno, indeksirano in ga je možno spreminjati. Omogoča enake elemente.

• **Tuple (terka)** je polje, ki je urejeno, indeksirano in ga ni možno spreminjati. Omogoča enake elemente.

• **Set (množica)** je polje, ki ni urejeno in ne indeksirano. Ne omogoča enakih elementov.

• **Dictionary (slovar)** je polje, ki ni urejeno, indeksirano in ga je možno spreminjati. Ne omogoča enakih elementov.

Primer: Uporaba List strukture.

```
List_f = ["apple", "banana", "cherry"]
print(List_f) #Izpis celega Lista-a

print(List_f [1]) #Izpis drugega elementa iz lista-a

List_f [0]= "mango" #Preimenovanje prvega elementa lista-a

List_f.append("kiwi" )   #dodajanje novega elementa v list

print(List_f [1:])  #izpis od drugega indeksa do konca >> banana,cherry

print(List_f [0: 1])  #izpis od drugega indeksa do konca >> apple, banana

print(List_f [-1: 1])  #izpis od drugega indeksa do konca >> cherry, apple, banana
```

Primer: Uporaba tuple strukture.

```
Tuple_f = ("apple", "banana", "cherry")
print(Tuple_f) #Izpis celega Tupla

print(List_f [1]) #Izpis drugega elementa iz lista-a

#Pretvorba tuple v list in nazaj v tuple

x = ("apple", "banana", "cherry")
y = list(x) #Pretvorba tuple v list
y[1] = "kiwi"
x = tuple(y) #Pretvorba list v tuple
```

Primer: Uporaba seta.

```
Set_f = {"apple", "banana", "cherry"}
print(Set_f)

Set_f.add("orange") #Dodajanje elementa v Set

print("banana" in Set_f) #Izpiši True, če je v Set-u element banana, sicer False
```

Primer: Uporaba slovarja (Dictionary).

```
Dict_car={
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(Dict_car)

x = Dict_car ["model"] #v x vrne Mustang

x = thisdict.get("model")#v x vrne Mustang

Dict_car ["year"] = 2020 #sprememba elementa
```

4. Zanke

Python skriptni jezik uporablja dva tipa zank: *for* in *while*. Nadalje si poglejmo način uporabe obeh tipov zank. V zanke lahko vgnezdimo poljubno število skriptnih ukazov.

Primer: Uporaba preproste python skripte ter uporaba for zanke

```
a=10     #Definirano celo število
b=12.34  #Definirano decimalno število
print("Hello WORD! ","\nIzpis stevil: ", a , b, "\nKonec izpisa!")

#1. FOR ZANKA – izpis elemenotv iz polja
print("\nFOR 1: Izpis elemenotv polja")
sadje = ["banana", "jagoda", "malina"]
for x in sadje:
  print(x)

#2. FOR ZANKA – izpis znak po znak
print("\nFOR 2: Izpis znak po znak")
for x in "banana":
  print(x)

#3. FOR ZANKA – izpis indeksov polja sadje
print("\nFOR 3: Izpis indeksov polja")
dol=len(sadje)          #Število elemenotv polja
print("Dolzina: ",dol)

for x in range(dol):
  print(x)
```

Primer 2: Uporaba while zanke.

```
# Izpis 0,1,2,3,4
print("1. While zanka - break")
st = 0
while True:
    print(st)
    st += 1
    if st >= 5:
        break    #Prekini zanko


# Uporaba ukaza else
print("\n2. While zanka - uporaba else")
st=0
while st<5:
    print(st)
    st +=1
else:
    print("Dosegli smo število ", st)


# Izpis lihih stevil -na intervalu od 0-15
print("\n2. FOR zanka - continue")
for x in range(15):
    # Preverimo če je število deljivo z 2
    if x % 2 == 0:
        continue   #Preskočimo izpis
    print(x)
```

5. Odločitveni stavek if-elif-else

Podobno, kot drugi programski jeziki lahko Python odločitvene pogoje podaljša preko 'elif'(else if) in 'else.' Prav tako omogoča poljubno vgnezdenje zank ter odločitvenih stavkov.

Primer: Uporaba različnih načinov pogojnega stavka *if*

```
#Pogojni stavek if-else
print("1. Pogojni stavek if-else")
a = 2
b = 3
if b > a:
    print("b je vecji od a")
else:
    print("a je vecji od b")

#Pogojni stavek if-elif-else
print("\n2. Pogojni stavek if-elif-else")
if b > a:
  print("b je večji od a")
elif a == b:
  print("a in b sta enaka")
else:
  print("a je vecji od b")


#Vgnezden pogojni stavek
print("\n3. Vgnezden Pogojni stavek")
if a > 1:
  print("a je vecji od 1")
  if a > 10:
    print("a je vecji tudi od 10")
  else:
    print("a ni vecji od 10")


#Kratki If stavek
print("\n4. Kratki if stavek")
print("a") if a > b else print("b")
print("a") if a > b else print("=") if a == b else print("b")
```

6. Vpis v spremenljivko

Uporabnik lahko vnese vrednost v poljubno spremenljivko z ukazom 'input'. Input ukaz formatira vrednost kot niz znakov (string). Če torej želimo vpisati številsko vrednost, moramo niz/string spremeniti v število s spreminjanjem podatkovnega tipa *'data type casting'*.

Primer: Vpis v spremenljivko

```
#Vpis v niz
ime = input("Vpisi ime:")
print("Hello ", ime)

#Vpis stevila in pretvorba podatkovnega tipa string-float
a=input("Vpisi stevilo: ")


b=float(a)+20; #float(2)-string v float stevilo, int(a)-string v celo stevilo
print("Stevilo: ",b)
```

7. Uporaba Python paketov in knjižnic

Nameščene pakete 'PIP' vključimo v skripto z ukazom `import` ter imenom knjižnice, podaljšek `as` pomeni, da med uporabo paketa uporabljamo lastno ime za dotični paket. V spodnjem primeru paket `matplotlib.pyplot` preimenujemo krajše v `plt` in `numpy` v `np`.

Primer: Uporaba knjižnice *numpy* in *matplotlib*

```
import matplotlib.pyplot as plt
import numpy as np

#Podrocje izpisa vrednost 'x'
x = np.arange(0, 2*np.pi,  0.1)   # start,stop,korak
dol=len(x)   #Dolzina polja 'x'
print(dol)

#Izracun sinusa za vsako točko
ysin = np.sin(x)
ycos =np.cos(x)

#Risanje funkcije
plt.plot(x,ysin, label="sine")
plt.plot(x,ycos, label="cosine")
plt.xlabel("$\omega$t")
plt.ylabel("Funkcije")
plt.legend()
plt.show()
```

8. I/O funkcije in upravljanje datotek

Za upravljanje ter kreiranje datotek uporabljamo open, read, write ter close. Način odpiranja datotek je podoben kot pri programskem jeziku C/C++. Način w – pisanje, r – branje, a – dodajanje podatkov.

Primer: Kreiranje datoteke ter vpis

```
#Kreiranje datoteke in vpis
a=10.34;
f = open("Moja_datoteka.txt", "w") #Nacin w-write, r-read, a-append,
f.write("Prav vrstica\n")

str= "Druga vrstica " + str(a)  #Sestavljen string s številom a
f.write(str)
f.close()
```

9. Try-except-finally

Funkcija je uporabna za testiranje parametrov skripte, kjer preko ukaza `try` preverimo obstoj le-te. Če ukaz `try` ne uspe izvršiti ukaza, se sproži ukaz `except`. Na koncu poizkusa `try - except` se vedno izvede ukaz `finally`, ki se pogosto uporabi za nadzor celotne skripte. V primeru, da je prišlo do napake ali izvedbe določenega segmenta skripte preko `try - except`, `finally`, lahko parameter vrne, kje je prišlo do izvršitev ali napake v skripti.

Primer: Uporaba try-except-finally

```
# Try - except - finally
try:    #Poizkusi izpisati d
  print(d)
except:
  print("d ne obstaja!") #V primeru, da spremenljivka d ne obstaja javi izpis
finally:
  print("Poizkus izpisa d- se je izvedel!") #Konec


# Try - except - finally v pri odpiranju datoteke

try:    #Poizkusi odpreti datoteko Moja_datoteka.txt
  f = open("Moja_datoteka.txt")
  f.write("Prva vrstica datoteke")
except:
  print("Prišlo je do težave pri vpisu v datoteko") #V primeru, da datoteka ne
obstaja
finally:
  f.close() #Zapri datoteko- ta ukaz se vedno izvede
  print("datoteko smo zaprli!") #Konec
```

10. Podprogrami in funkcije

Da zagotovimo boljši pregled ter strukturo skripte, je smiselna uporaba podprogramov. Podprogram služi, da določen del kode združimo pod enim imenom. Vsak podprogram v Pythonu ima vhodne in izhodne argumente, ki jih določi uporabnik. Prav tako lahko podprograme ločimo v svojo ločeno skripto. Podprogram definiramo z ukazom:

```
def ime_funkcije(vhodni argumenti):
  skriptni ukazi
  return izhod
```

Za primer bomo pokazali tudi način uporabe glavnega program 'main', ki pa ni nujno potreben za izvajanje skripte.

Primer: Uporaba podprograma v isti skripti

```
### Podprogram 1
def Podprogram_1(vhod,num):
  for x in vhod:   #Izpis polja
    print(x)
  return 2*num  #Vrednost vrnemo nazaj

### Podprogram 2
def Podprogram_2(vhod,index):
   print(vhod[index])#Izpis polja
#### Konec Podprograma 1 in 2


#Spremenljivke
sadje = ["banana", "jagoda", "malina"]
b=12.3;


#Klic podprograma, argumenta: sadje in b
print("Klic podprograma 1 iz skripte extern_fun: ");
a=Podprogram_1(sadje,b)
print("Izpis ",a);

#Klic podprograma, argumenta: sadje in index polja
print("\nKlic podprograma 2 iz skripte extern_fun: ");
Podprogram_2(sadje,2)   #Izpisemo tretji element polja
```

Primer: Uporaba podprogramov, ki so ločeni v drugi skripti z imenom **extern_fun.py**.
Glavna skripta:

```
import extern_fun as f #Dodajanje zunanje skrpite extern_fun kot ‘f’

#Spremenljivke
sadje = ["banana", "jagoda", "malina"]
b=12.3;

#Klic podprograma, argumenta: sadje in b
print("Klic podprograma 1 iz skripte extern_fun: ");
a=f.Podprogram_1(sadje,b)
print("Izpis ",a);

#Klic podprograma, argumenta: sadje in index polja
print("\nKlic podprograma 2 iz skripte extern_fun: ");
```

```
f.Podprogram_2(sadje,2)   #Izpisemo tretji element polja


#Klic podprograma, argumenta: z večimi izhodi
print("\nKlic podprograma 3 iz skripte extern_fun: ");
d=f.Podprogram_3(sadje,2)   #Funkcija v spremenljivko d vrne polje-sadje in b
print("Sadje:",d[0][0]," ",d[0][1]," ",d[0][2]);
print("Stevilo ",d[1]);
```

Skripta s podprogrami z imenom '**extern_fun.py**'. Skripta vsebuje tri podprograme z imeni 'Podprogram_1', 'Podprogram_2' in 'Podprogram_3'.

```
### Podprogram 1
def Podprogram_1(vhod,num):
  for x in vhod:   #Izpis polja
    print(x)

  return 2*num  #Vrednost vrnemo nazaj

### Podprogram 2
def Podprogram_2(vhod,index):
    print(vhod[index])#Izpis polja

### Podprogram 3
def Podprogram_3(vhod,num):
  for x in vhod:   #Izpis polja
    print(x)

  vhod=['češnje','slive','hruške'];
  b=2.2*num
  return [vhod,b]  #Na izhodu imamo polje-'vhod' in enojno spremenljiko 'b'
```

Python programski jezik omogoča tudi uporabo glavne funkcije main(). Uporaba funkcije main() pomeni, da se skripta začne izvajati iz glavnega programa.

Primer: Uporaba `main()` glavnega programa, kjer vključimo zunanjo skripto '**extern_fun.py**'.

```
import extern_fun as f #Dodajanje zunanje skrpite

b=12.3; #Globalna spremenljivka

#Glavni program
def main():
    print("Hello from MAIN function!")
    sadje = ["banana", "jagoda", "malina"]
    global b #Vpeljemo globalno spremenljivko

    #Klic podprograma, argumenta: sadje in b
    print("\nKlic podprograma 1 iz skripte extern_fun kot 'f': ");
    a=f.Podprogram_1(sadje,b)
    print("Izpis ",a);

    #Klic podprograma, argumenta: sadje in index polja
    print("\nKlic podprograma 2 iz skripte extern_fun kot 'f': ");
    f.Podprogram_2(sadje,0)   #Izpisemo tretji element polja


#Deklaracija glavnega programa
if __name__ == "__main__":
    main() #Klic glavnega programa
```

Lambda funkcija – posebna kratka anonimna funkcija. Anonimna pomeni, da nima imena in se uporablja samo v dotični vrstici Python skripte.

Primer: Lambda funkcija, kjer je x lambda funkcija z enim argumentom.

```
lambda argument : izraz

#Lambda funkcija z enojnim argumentom

 x = lambda b : b - 2
print(x(5)) #Izpis je 3 (5-2)


#Lambda funkcija s trojinm argumentom

x = lambda a, b, c : a + b - c
print(x(2, 3, 1)) #Izpis je 4 (2+3-1)


#Lambda funkcija v podprogramu Fun_1

def Fun_1(b):
  return lambda a : a + b

a= Fun_1(2) #Vnos za vrednost b

print(a(5)) #Vnos za vrednost a, Izpis 7 (5 + 2)
```

11. Serijski vmesnik

Serijski vmesniki je osnovna periferna naprava mnogih mikrokrmilniških sistemov, senzorjev in indikatorjev. Serijski vmesnik je v osnovni sestavljen iz RX- in TX-linije, po katerih se serijsko prenašajo podatki po paketih različnih bitnih dolžin. Serijski vmesnik pozna še druge kontrolne linije, ki so namenjeni za nadzor prenosa podatkov, a jih tukaj ne bomo posebej obravnavali. Najpogostejše dolžine paketov so 7,8 ali 9 bitov. V jeziku Python je branje serijskega vmesnika omogočeno preko knjižnice *serial*.

Primer: Branje serijskega vmesnika na portu COM 3 in bitno hitrost 115200bit/s

```
import serial

ser = serial.Serial("COM3", 115200)

while True:
     read_serial=str(ser.readline())  #Branje iz porta COM 3
     print("Prejeti podatki: ",read_serial);
```

Primer: Naprednejša nastavitev serijskega vmesnika na portu COM 3, bitno hitrostjo 38400bit/s, brez paritete, enim stop bitom, velikostjo paketa 8bitov in brez ' `timeout` ' časovnika.

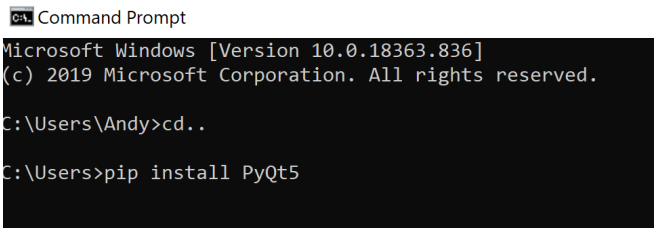
```
import serial

#Nastavitev serisjekga porta
s = serial.Serial(port='COM3',baudrate=38400,parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,\
                    bytesize=serial.EIGHTBITS, timeout=0)


#Shranjevanje podtakov
buffer = []

while True:
    for c in s.read():   #beremo znak po znak
        buffer.append(c) #prebrani znak dodamo v buffer
        if c == '\n':
            print("prejeto: ", buffer) #izpiši vrstico
            buffer = []  #pobriši buffer
            break

s.close() #zapri serijski vmesnik COM3
```

12. Namestitev pyQT5

Za načrtovanje grafičnega vmesnika bomo uporabili knjižnico pyQt5, ki omogoča snovanje grafičnih vmesnikov v Python skriptnem jeziku. PyQT vsebuje bogat nabor orodij in elementov, s katerimi lahko načrtujemo različne tipe vmesnikov. Za lažjo namestitev različnih Python inštalacijskih paketov pip, omogočimo operacijskemu sistemu, da upravlja s Python interpreterjem, tako da zaženemo skripto **win_add2path.py**, ki se nahaja na lokaciji (`\\Python37\Tools\scripts\win_add2path.py`), ali (`\\Program Files (x86)\Microsoft Visual Studio\Shared`).

PyQT5 namestimo tako, da v komandnem oknu 'cmd' zaženemo ukaz (https://pypi.org/project/PyQt5/):

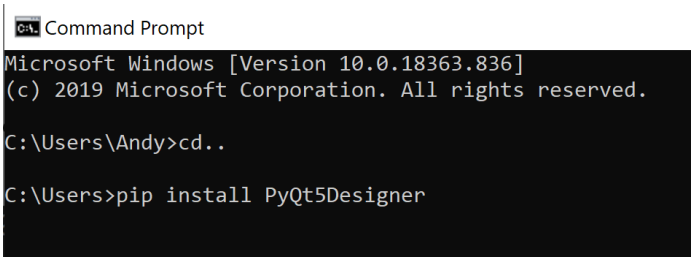
```
pip install PyQt5
ali
py -m pip install PyQt5
```



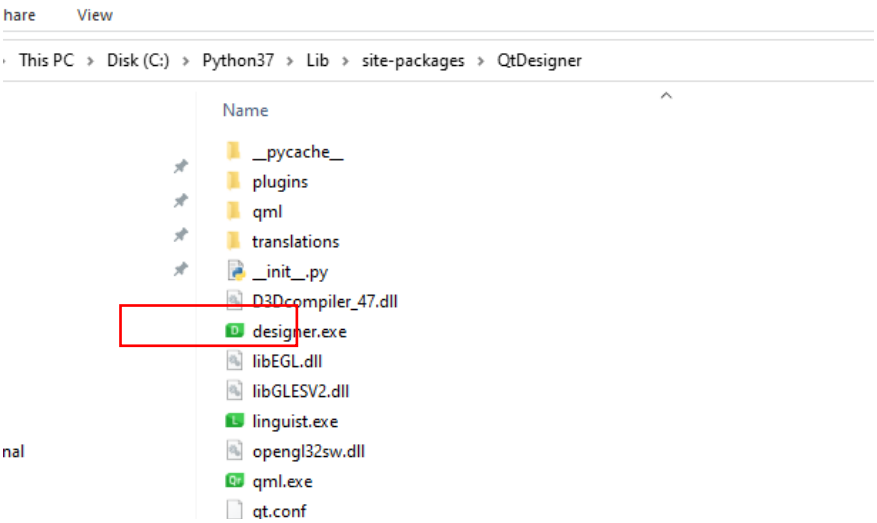
Namesto `pip install PyQT5` lahko uporabimo ukaz `py -m pip install PyQT5`.

Dodatno namestimo tudi grafični urejevalnik PyQTDesigner, ki nam omogoča grafično postavitev elementov grafičnega vmesnika. PyQT5 designer namestimo z ukazom (https://pypi.org/project/PyQt5Designer/):

```
pip install PyQt5Designer
ali
py -m pip install PyQt5Designer
```



Po uspešni namestitvi grafični urejevalnik PyQT5Designer najdemo v datoteki (`\\Python37\Lib\site-packages\QtDesigner`).



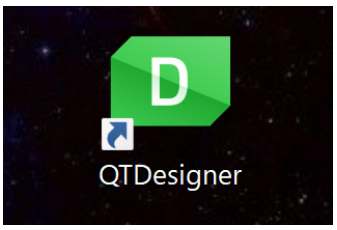
Za izrisovanje grafov namestimo napredno knjižnico ***pyqtgraph***, ki nudi mnoga orodja za prikaz podatkov realnega časa ali statičnih grafov.

```
pip install pyqtgraph
```
ali
```
py -m pip install pyqtgraph
```

Knjižnice lahko namestimo tudi ročno, z direktno namestitvijo binarnega kolesa (ang. ***Binary wheel*** s konjčnico `.whl`), kjer navedemo različico Pythona (v našem primeru 3.7) ter ime binarnega kolesa, ki ga skopiramo v datoteko, kjer se nahaja Python3.X.

```
py -3.7 -m pip install PyQt5-5.14.2-5.14.2-cp35.cp36.cp37.cp38-none-win_amd64.whl
```

S kreiranjem grafičnega vmesnika pričnemo z zagonom programa designer.exe.



Najpogostejši elementi GUI-a:

QPushButton – gumb,
QLabel – besedilo,
QLineEdit – vnos besedila,
QListWidget – prikaz besedila polju.

Generiranje Python skripte iz .ui objekta

Po končanem grafičnem snovanju grafičnega vmesnika, kjer vmesnik shranimo s končnico '.ui', funkcionalnost elementom vmesnika dodajmo v novi Python skripti, kjer posebej uvozimo datoteko '.ui'.

V naše primeru bomo uporabili skripto, kjer je imamo grafični vmesnik shranjen kot 'my_gui.ui' in ime razreda je QMainWindow.

Primer: Skripta z vpeljavo '.ui' datoteke.

```
from PyQt5 import QtWidgets, uic
import sys
from multiprocessing import Process
import threading


# GUI class
class Ui(QtWidgets.QMainWindow):# CLASS NAME: QMainWindow
    def __init__(self):
        super(Ui, self).__init__()
        uic.loadUi('my_gui.ui', self)  # GUI file from QTdesigner

        #Set GUI title
        self.setWindowTitle("My first Python app!")

        #Import elements in GUI
        self.button = self.findChild(QtWidgets.QPushButton, 'Button_Send') # Button
        self.button.clicked.connect(self.Button_function) # function to Button

        self.lable_status = self.findChild(QtWidgets.QLabel, 'label_status') #Label
        self.list = self.findChild(QtWidgets.QListWidget, 'listWidget') #QlistWidget

        self.input_text = self.findChild(QtWidgets.QLineEdit, 'lineEdit') #QLineEdit

        #Show GUI
        self.show()

    #Button function on click
    def Button_function(self):
          self.button.setText("OFF")  #Change button text



#Main function
if __name__ == '__main__':

    #RUN GUI
    app = QtWidgets.QApplication(sys.argv)
    window = Ui() #Open GUI
    Process(target=app.exec()).start() #Start Mulithreading
```

C-jezik

13. Navodila za programiranje v programskem jeziku C

Program v programskem jeziku C je sestavljen iz naslednjih sklopov [5]:

- predprocesorski ukazi (#include, #define, #line, #ifdef…)
- deklaracija tipov
- spremenljivke
- funkcije

Vsak program v C-ju se prične s klicem funkcije **`main ()`**. Funkcija **`main`** je preprost skupek ukazov, ki se proži ob začetku programa. Funkcija **`main`** se v programu pojavi le enkrat, **`'main'`** se prične z oklepajem in zaklepajem in je znak za prevajalnik, da gre za glavno funkcijo.

Primer: Osnovna funkcija 'main'.

```
#include <stdio.h>

int main()
{
  return 0;
}
```

Dva zavita oklepaja definirata začetek in konec programa (vsebino funkcije) in vse programske stavke postavimo med njiju. Ukaz '***return 0***' pomeni, da je program izveden uspešno in je definiran kot makro v knjižnici '***stdio.h***' , ki je vključen s predprocesorskim ukazom ***include***.

14. Komentarji

Komentarji v kodi so sestavnimi del vsakega programa. Komentarji služijo za lažje razumevanje segmentov programa in ne vplivajo na potek in izvedbo programa [7]. Komentarji niso ukazi ali sam program! Komentarje pišemo v obliki:

// komentar ali */* komentar*/*

Če uporabimo način //, pomeni, da se komentar zaključi na koncu vrstice. Če uporabimo način */* komentar */*, se komentar prične s simboloma /* in se nadaljuje tako dolgo, dokler ga ne zaključimo s simboloma */. Ta način komentarja imenujemo tudi večvrstični komentar.

Primer: Uporaba komentarja v programu

```
#include <stdio.h>
// moj prvi program
/* program ne
   naredi nicesar */
int main()
{
	return 0; //Konec programa
}
```

15. Deklaracija spremenljivk

Pri uporabi spremenljivk v programskem jeziku C je potrebno definirati imena spremenljivk. Imena spremenljivk definiramo tako, da upoštevamo naslednja pravila [6]:

- Ime spremenljivke lahko vsebuje črke, števila in podčrtaje.
- Priporočljivo je, da se spremenljivka prične s črko in ne z drugim znakom, kot je številka ali podčrtaj.
- Za ime spremenljivke ne smemo izbrati rezerviranih besed.

Tipi spremenljivk:

Tabela 11: Tipi spremenljivk v programskem jeziku C.

Tip spremenljivke	Beseda	Število bajtov	Območje
Karakter	**char**	1	-128 do 127
Celo število	**int**	2/4	-32768 do 32767
Kratko celo število	**short**	2	-32768 do 32767
Dolgo celo število	**long**	8	+- 2.147.483.64(8/7)
Nepredznačen karakter	**unsigned char**	1	0 do 255
Nepredznačeno celo število	**unsigned int**	2/4	0 do 65535
Nepredznačeno kratko celo število	**unsigned short**	2	0 do 65535
Nepredznačeno dolgo celo število	**unsigned long**	8	0 do 4 E9
Plavajoča vejica enojna natančnost	**float**	4	1.2E-38 to 3.4E+38
Plavajoča vejica dvojna natančnost	**double**	8	2.3E-308 to 1.7E+308
Plavajoča vejica najvišja natančnost	**long double**	10	3.4E-4932 to 1.1E+4932

Primer: Deklaracija spremenljivke

```
#include <stdio.h>

int main()
{
	int a, b, c, d = 2;  // deklaracija celih števil
	char g=A;     // deklaracija karakterja
	a=4;
	b = 0; c = 2;
	return 0;
}
```

16. Binarni matematični operatorji ter vhodno-izhodne funkcije

Tabela 12: Matematični operatorji v C-ju.

Operator	Simbol	Primer
vsota	+	x + y
razlika	-	x - y
množenje	*	x * y
deljenje	/	x / y
ostanek	%	x%y

Enostavni matematični operatorji

Tabela 13: Enostavni matematični operatorji v C-ju.

Operator	Simbol	Primer
povečanje	++	++x , x++
zmanjšanje	--	--x, x--

Pomen:

++x predponski način (povečaj za 1 in nato uporabi trenutno vrednost)

x++ priponski način (uporabi trenutno vrednost in nato povečaj za 1)

--x predponski način (zmanjšaj za 1 in nato uporabi trenutno vrednost)

x-- priponski način (uporabi trenutno vrednost in nato zmanjšaj za 1)

x=+1 je enako x = x+1

x=-1 je enako x = x -1

Relacijski operatorji

Tabela 14: Ralcijski operatorji.

Operator	Simbol	Primer
enakost	==	x == y
večje	>	x > y
manjše	<	x < y
večji ali enak	>=	x >= y
manjši ali enak	<=	x <= y
neenakost	!=	x != y

Logični operatorji

Tabela 15: Logični operatorji.

Operator	Simbol	Primer
IN	&&	x && y
ALI	||	x || y
NE	!=	x != y

Bitne logične operacije

Bitne logične operacije se vršijo nad vsakim istoležečim bitom spremenljivke tipa **int** ali **char**.

Tabela 16: Bitne logične operacije

Operator	Simbol	Primer
IN	&	x & y
ALI	|	x | y

Izjavnostna algebra logičnih operacij:

Tabela 17: Izjavnostna tabela

A	B	ALI	IN	NE A	NE B	XAND	XOR
0	0	0	0	1	1	0	0
0	1	1	0	1	0	0	1
1	0	1	0	0	1	0	1
1	1	1	1	0	0	1	0

Ubežne sekvence

Ubežne sekvence določajo način izpisa besedila na zaslon.

\a – pisk
\b – vzvratna tipka
\n – nova vrstica
\t – tabulator
\\ - poševnica backslash (\)

Konverzacijske označbe

Tabela 18: Konverzacijske označbe v C-ju.

Označba	Pomen	Tip spremenljivke
%c	karakter	**char**
%d	celo število	**int, short**
%ld	dolgo celo število	**long**
%f	decimalno število	**float, double**
%s	niz (string)	**polje karakterjev**
%u	nepredznačeno celo število	**unsigned int, unsigned short**
%lu	nepredznačeno dolgo celo število	**unsigned long**
%o	osmiška notacija	**int, short**
%x	**šestnajstiška notacija**	

Funkcija za vpis in izpis

Ukaza za vpis in izpis sta:
scanf – funkcija odčita/prebere vrednost vhodne spremenljivke
printf – funkcija na zaslon izpiše besedilo ali/in vrednosti spremenljivk

Primer: Izpis besedila.

```
#include <stdio.h>

int main()
{
	printf("To je prvi izpis besedila.\n");  // izpis besedila na zaslon
	return 0;

}
```

Primer: Izpisa spremenljivke.

Spremenljivke izpišemo v funkciji *printf* s pomočjo konverzacijske označbe, katere vključimo v funkcijo *printf* kot dodatni argument.

```
int main()
{
	int a = 1;    // deklaracija spremenljivke a
	float b = 5.2;

	printf("%d", a);     //izpis vrednosti spremenljivke a
	printf("%f", d);     //Izpis vrednosti decimalnega stevila b
	printf("Stevilo a=%d /n", a); //izpis besedila in vrednosti spremenljivke a

	return 0;

}
```

Primer: Branje vhodnih spremenljivk

```
int main()
{
	int a, b, c;

	printf("Vpisite poljubno celo pozitivno stevilo");
	scanf("%d", &a);     //branje vhodne spremenljivke

	printf("Vpisite dve poljubni celi stevili.");
	scanf("%d %d", &b, &c);

	printf("Vnesli ste stevila: %d %d %d", a, b, c);

	return 0;
}
```

Primer: Branje vhodnih spremenljivk in pomik izpisa

```
int main()
{
	int a, b;
```

```
        float d = 7.13455; b = 3;

        printf("Vpisite poljubno celo pozitivno stevilo");
        scanf("%d", &a);     //branje vhodne spremenljivke

        //Pomik decimalnega stevila za vrednost spremenljivke a
        printf("Vnesli ste stevilo: %*f", a, d);

        // Izpis decimalnega stveila z enim decimalnim mestom
        printf("Vnesli ste stevilo: %.1f \n", d);

        // Izpis decimalnega stevila f z b decimalnimi mesti in pomikom a
        printf("Vnesli ste stevilo: %*.*f \n", a, b, d);

        return 0;
}
```

17. Zanke

Programski jezik C ima več načinov za tvorbo zank in pogojnih stavkov. Zanka se izvaja tako dolgo, dokler je pogoj izpolnjen. Zanka lahko vsebuj vse funkcije, ki so deklarirane v programskem jeziku C.

WHILE zanka

Program vstopi v zanko, ko je izpolnjen pogoj. Zanka se izvaja tako dolgo, dokler je pogoj izpolnjen. Struktura *while* zanke:

```
while (pogoj)
{
    operacije;
}
```

Primer: Uporaba *while* zanke

```
#include <stdio.h>

int main()
{
        int stevilo;

        stevilo = 0;  // zacetna vrednost deklariranega stevila

        while (stevilo < 4)  // pogoj zanke while
        {
                printf("Trenutna vrednost stevila %d\n", stevilo);

                stevilo = stevilo + 1;      //inkrement stevila za 1
        }

        return 0;
}
```

DO WHILE zanka

Funkcije znotraj zanke se izvedejo vsaj enkrat in se izvajajo tako dolgo, dokler je pogoj zanke izpolnjen. Struktur zanke ***do while***:

```
do
{
    operacije;

} while (pogoj);
```

Primer: Uporaba *do while* zanke

```
#include <stdio.h>

int main()
{

	int stevilo=0;


	//Do while
	do
	{
		printf("Trenutna vrednost stevila %d\n", stevilo);
		stevilo = stevilo + 1;

	} while (stevilo < 4);

	return 0;

}
```

FOR zanka

For zanka se izvaja tako dolgo, dokler je pogoj izpolnjen. Sintaksa ***for*** zanke zahteva vnos začetnega stanja, končnega stanja ter koraka. Struktura ***for*** zanke:

```
for (zacetno stanje; koncno stanje; korak)
{
    operacija;
}
```

Primer: Uporaba *for* zanke

```
#include <stdio.h>

int main()
{
	int stevilo;

	for (stevilo = 0; stevilo < 4; stevilo++)
	{
		printf("Trenutna vrednost stevila %d\n", stevilo);
	}

	return 0;
}
```

18. Odločitveni stavki

IF odločitveni stavek

Odločitveni stavek preveri pogoj in izvrši ukaz, če je pogoj pravilen. V primerjavi z zankami, se pri stavku ***if*** operacija izvede le enkrat in se ne ponavlja.

Struktura *if* stavka z enim pogojem:

```
if (pogoj)
operacija;

ali

if (pogoj)
{
     operacija1; operacija2;
}
```

Vgnezdenje ukazov v *if* odločitveni stavek:

```
if (pogoj)
{
    operacija1; operacija2;
}

ali

if (pogoj)
operacija1; operacija2;
```

Uporaba *else* in *else if*

Uporaba ***else*** in ***else if*** stavka omogoča podaljšanje odločitvenega stavka, če prvotni pogoj v deklaraciji ***if*** ni izpolnjen. Struktura stavka ***if*** in ***else if***.

```
if (pogoj1)
operacija1;
else
operacija2;

ali

if (pogoj1)
```

```
operacija1;
else if (pogoj2)
operacija2;
else
operacija3;
```

Stavek *else if* je možno uporabiti večkrat zaporedoma.

```
if (pogoj1)
operacija1;      //operacija1 se izvede, če je pogoj1 izpolnjen
else if (pogoj2)
operacija2;      //operacija2 se izvede, če je pogoj2 izpolnjen
else if (pogoj3)
operacija3;      //operacija3 se izvede, če je pogoj3 izpolnjen
else
operacija4;      //operacija4 se izvede, če ni izpolnjen noben
prejšnji pogoj
```

Primer: Uporaba odločitvenih stavkov *if, else if* in *else.*

```
#include <stdio.h>

int main()
{
	int a; a = 2;

	if (a == 2)
	{
		printf("Vrednost stevila je %d\n", a);

	}else if (a < 4)
	{
		printf("Vrednost stevila je %d kar je manj od stevila 4.\n", a);

	}else
	{
		printf("Vrednost stevila je %d kar je vec od stevila 4.\n", a);
	}

 return 0;
}
```

SWITCH odločitveni stavek

Switch stavek je podoben zaporedju ***if, else if, else*** stavkov na preverjanju pogoja ene spremenljivke.

Struktura ***switch*** stavka:

```
switch (spremenljivka)
{
    case vrednost1:
         operacija1;
         break;
    case vrednost2:
         operacija2;
            break;
    case vrednost3:
         operacija3;
         break;
    default:
         operacija4;
         break;
}
```

Izraz ***default*** se lahko vključi v primeru, če noben pogoj-***case*** znotraj ***switch***-a ni izpolnjen.

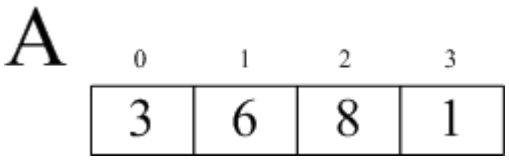
Primer: Uporaba odločitvenega *switch* stavka

```
#include <stdio.h>

int main()
{
       int a = 10;

       switch (a)
       {
       case 10:
              printf("Stevilo a je 10. \n");
              break;

       case 20:
              printf("Stevilo a je 20. \n");
              break;

       default:
              printf("Stevilo a ni enako 10 ali 20. \n");
              break;
       }

 return 0;
}
```

19. Polja

Polje je serija homogenih delov, ki so deklarirani kot isti podatkovni tip. Poznamo več vrst polj glede na dimenzijo (enodimenzionalna, dvodimenzionalna itd.). Polje vsebuje celice, v katerih so shranjene vrednosti spremenljivk. Vsaka celica v polju ima naslov ali index. Če operiramo z enodimenzionalnim poljem, potem naslov vsebuje le en podatek. Indeksiranje celic se v programskem jeziku C prične z 0.

Primer: Enodimenzionalno polje



Kličemo celice *A[0]=3; A[1]=6; A[2]=8; A[3]=1*.

Primer: Dvodimenzionalno polje

indeks	0	1	2	3
0	3	6	8	1
1	56	23	89	9
2	90	100	0	55

Kličemo vrednosti celice po stolpcu: *A[0,0]=3; A[1,0]=56; A[2,0]=90;*
Kličemo vrednosti celice po vrstici*: A[1,0]=56; A[1,1]=23; A[1,2]=89; A[1,3]=9;*

Deklaracija polja

Polje deklariramo enako kot spremenljivko, tako da ji najprej določimo podatkovni tip (*int, double, float, char...),* nato ime in potem dimenzijo oziroma velikost polja.

Primer: Deklaracija polja

```
#include <stdio.h>

int main()
{
	int polje1[3];		// enodimenzionalno polje velikosti 3
```

```
	int polje2[3][4] = { {3,6,8,1},{56,23,89,9},{90,100,0,55} };

	polje1[0] = 1;		// zapis vrednosti v polje v prvo celico
	polje1[1] = 2;	// zapis vrednosti v polje v drugo celico
	polje1[2] = 3;	// zapis vrednosti v polje v tretjo celico

	return 0;
}
```

Izpis polja

Izpis celotnega polja ni mogoč z enim ukazom, kot je to mogoče pri navadnih spremenljivkah. Programski jezik C omogoča možnost izpisa ene spremenljivke, zato polje izpišemo s pomočjo zank, pri čemer vsako celico izpišemo posebej. Glede na dimenzijo polja uporabimo število zank. Če je polje enodimenzionalno, uporabimo le eno zanko, za dvodimenzionalno uporabimo dve zanki in tako naprej.

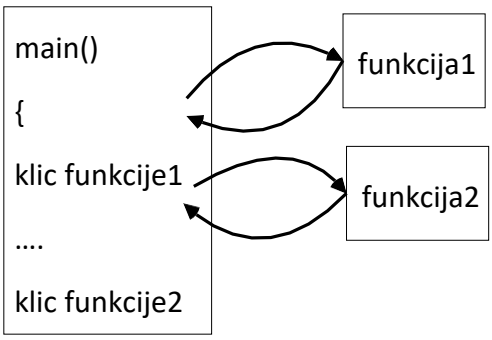
Primer: Izpis polja

```
#include <stdio.h>
int main()
{
	int i, j;

	// enodimenzionalno polje velikosti 3
	int polje1[] = { 2,3,4 };

	// dvodimenzionalno polje velikosti 3 x 4
	int polje2[][] = { {3,6,8,1},{56,23,89,9},{90,100,0,55} };


	// Izpis enodimenzionalnega polja
	for (i = 0; i <= 2; i++)
	{
		printf("%d \t", polje1[i]);
	}

	// Izpis dvodimenzionalnega polja
	for (i = 0; i <= 2; i++)	//zanka za vrstico
	{
		for (j = 0; j <= 3; j++) //zanka za stolpec
		{
			printf("%d \t", polje2[i][j]);
		}
	}
}
```

20. Funkcije – podprogrami – prototipi

Funkcije predstavljajo neodvisni del kode in izvajajo del programa, ki se kliče iz glavnega programa ali druge funkcije. Tukaj imajo pomen globalne in lokalne spremenljivke. Lokalne spremenljivke veljajo samo znotraj funkcije ali glavnega programa, kjer so deklarirane (lokalni nivo). Globalne spremenljivke veljajo povsod, tako znotraj glavnega programa ***main*** kakor izven njega. V tem primeru

so deklarirane izven funkcij na začetku programa (za zaglavjem *include*).



Deklaracija funkcije

Funkciji najprej določimo podatkovni tip, nato njeno ime ter vhodne spremenljivke (parametre), ki jih bomo prenesli iz klicanega dela programa. Podatke funkcije vrnemo z ukazom ***return***. V programu funkcijo kličemo po imenu, s katerim je deklarirana.

```
tip imeFunkcije(tip1 spremenljivka1, tip2 spremenljivka2)
{
     tip3 spremenljivka3;
     spremenljivka3 = spremenljivka1 + spremenljivka2;
     return spremenljivka3;
}
```

Pomni! Podatkovni tip funkcije določa tip podatka, katerega vrne ukaz ***return***. Tip funkcije *'imeFunkcije'* in tip spremenljivke v ***return*** *spremenljivka3* sta enaka (*tip* funkcije je enak *tip3*)!

Primer: Glavni program in klic funkcije

```
#include <stdio.h>

int vsota(int a, int b) // Zapis funkcije vsota
{
       return (a + b);
}

int main()    //Zacetek glavnega programa
{
       int a = 1, b = 3, rezultat;

       printf("Vpisite dve poljubni celi stevili /n");
       scanf("%d %d", &a, &b);

       //vrednost, ki jo vrne funkcija se priredi v spremenljivko rezultat
       rezultat = vsota(a, b);     //Klic funkcije vsota
       printf("Vsota je = %d \n", rezultat);

    return 0;
}
```

Program je zapisan z uporabo lokalnih spremenljivk. (a,b,rezultat)

Primer: Glavni program in klic funkcije z globalnimi spremenljivkami

```
#include <stdio.h>

int a, b;     // Globalni spremenljivki

// Zapis funkcije vsota
int vsota()
{
       int vrednost; // deklaracija lokalne spremenljivke vrednost
       vrednost = a + b;
       return vrednost;
}


int main()    // Pricetek glavnega programa
{
       int rezultat; // lokalna spremenljivka v funkciji main

       printf("Vpisite dve poljubni celi stevili \n");
       scanf("%d %d", &a, &b);

       rezultat = vsota(a, b); //Klic funkcije vsota
       printf("Vsota je = %d \n", rezultat);

       return 0;
}
```

V tem primeru ni bilo potrebno klicati vhodne spremenljivke znotraj funkcije, ker so bile definirane zunaj glavnega programa ***main()*** kot globalne spremenljivke.

Pomni! Globalne spremenljivke imajo doseg skozi celoten program in se deklarirajo samo enkrat v programu (načeloma na začetku). Lokalne spremenljivke so dosegljive samo znotraj funkcije, kjer so deklarirane. To pomeni, da lahko v različnih funkcijah deklariramo lokalne spremenljivke z enakim imenom.

21. Niz (string)

Niz znakov ali string predstavlja polje znakov. Deklariramo na podoben način kot polja, le da določimo podatkovni tip *char*. Za uporabo niza znakov v programskem jeziku C je potrebno vključiti knjižnico *#include <string.h>*, ki vsebuje funkcije za operacije s nizi.

Primer `char string[10];`

Deklarirali smo podatkovni niz dolžine 10.

Funkcije za operacije z nizi

Tabela 19: Funkcije za operacije z nizi.

strlen(*string*)	vrne celo število, ki predstavlja dolžino niza brez ničel
strcpy(*string1,string2*)	kopira besedni niz ali celoten string2 v string1
strncpy(*string1,string2, dolzina*)	kopira besedni niz ali le del niza v drugi niz
strcat(*string1,string2*)	združevanje celotnih nizov 1 in 2
strncat(*string1,string2, mesto*)	združevanje nizov in določitev mesta združevanja
strcmp(*string1,string2*)	primerjava celotnih nizov rezultat 1 ali -1, *string1>string2=1; string1<string2=-1*
strncmp(*string1,string2. mesto*)	primerjava stringov 1 in 2 z možnostjo določanja mesta

Primer: Uporaba niza znakov – string.

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

int main()
{
	char ime1[12], ime2[12], skupaj[25]; //deklaracija nizov
	char naslov[20];

	strcpy(ime1, "Bojan"); //vpis vrednosti v niz ime
	printf("Vpisi poljubno ime");
	scanf("%s", ime2); //vpis imena v niz ime2

	//vpis vrednosti v niz naslov
	strcpy(naslov, "Uporaba podatkovnih nizev.");

	printf("	%s \n\n", naslov); //izpis vrednosti niza naslov printf("Ime 1 is
%s\n", ime1); //izpis vrednosti niza ime1 printf("Ime 2 is %s\n", ime2); //izpis
vrednosti niza ime2

	if (strcmp(ime1, ime2) > 0) /* vrne 1 če je ime1 > ime2 */
		strcpy(skupaj, ime1);	/* kopira ime1 v niz skupaj */
	else
		strcpy(skupaj, ime2);	//drugače kopira ime2 v skupaj

		//izpis vecjega imena
	printf("Vecje alfabeticno ime je %s\n", skupaj);

	strcpy(skupaj, ime1); //kopiramo niz ime1 v niz skupaj
	strcat(skupaj, "	");//kopiramo v niz skupaj presledek "	"
	//presledek uporabimo zato, da ne izpise imen skupaj

	strcat(skupaj, ime2); //nizu dodamo ime2
	//niz skupaj vsebuje niza ime1 in ime2

	//niz skupaj izpisemo na zaslon
	printf("Obe imeni sta %s\n", skupaj);

	return 0;
}
```

22. Strukture

Struktura je podatkovni tip, ki jo deklarira uporabnik in lahko vsebuje različne podatkovne tipe (int, short, long, float itd.). Struktura nam tako omogoča deklaracijo novega izpeljanega podatkovnega tipa, ki je kompleksnejši in preglednejši.

Deklaracija strukture:

```
struct imeStrukture1
{
     podatkovniTip spremenljivka1;
     podatkovniTip spremenljivka2;
     podatkovniTip spremenljivka3;
};
```

Strukture so lahko globalne ali lokalne, tako kot spremenljivke.

Klic podatkov v strukturi

Podatke v strukturi kličemo tako, da najprej zapišemo ime strukture in nato za piko navedemo element v strukturi, s katerim želimo operirati.

Primer: `struct imeStrukture1 objekt1;`

Klic se nato izvrši na naslednji način:

```
objekt1.spremenljivka1
objekt1.spremenljivka2
objekt1.spremenljivka3
```

Primer: Uporaba strukture za izračun površine kvadrata

```
#include <stdio.h>

struct podatki     //Deklaracija strukture
{
       float sirina; //Spremenljivka strukture tipa float
       float visina; //Spremenljivka strukture tipa float
};


int main()
{
       float povrsina;

       //deklaracija objekta p, ki ima enake lastnosti kot struktura podatki
       struct podatki p;

       //vpis vrednosti v spremenljivko sirina
       p.sirina = 2;
       //vpis vrednosti v spremenljivko visina
       p.visina = 3;

       povrsina = p.sirina * p.visina; //izracun povrsine
```

```
        printf("Povrsina je %0.2f \n\n", povrsina);

        return 0;

}
```

Strukturo lahko definiramo tudi kot polje, kar pomeni, da lahko obstaja več struktur z enako formo.

Primer: Struktura

```
struct podatki       //Deklaracija strukture
{
        float sirina; //Spremenljivka strukture
        float visina; //Spremenljivka structure
}
```

V tem primeru imamo definirano strukturo v polju velikosti 10. Klic takšne strukture je enak, le da upoštevamo naslove, ker operiramo s poljem.

Primer: Polje struktur

```
struct podatki p[3]; //deklaracija polja z imenom p (=polje objektov)

   p[0].sirina=50;          //objekt p z indekom 0
   p[0].visina = 50;

   p[1].sirina = 100;       //objekt p na indeksom 1
   p[1].visina=100

   p[2].sirina = 150;       //objekt p na indeksom 2
   p[2].visina=150;

   povrsina = p[0].sirina * p[0].visina;  //izracun za objekt z indeksom 0
```

Če ne želimo uporabljati polja objektov, ampak želimo uporabiti objekte z različnimi imeni in enako strukturo, to lahko definiramo kot:

Primer: Struktura več elementov

```
struct kvadrat, trikotnik, krog    //Deklaracija treh struktur
{
        float sirina; //Spremenljivke strukture
        float visina; //Spremenljivke strukture
};
```

Primer: Klic spremenljivk

```
// Deklaracija objektov z enakimi spremenljivkami struktur
struct trikotnik tr;
struct krog kr;

kv.sirina = 2; kv.visina = 2;
tr.sirina = 2; tr.visina = 4;
kr.sirina = 4; kr.visina = 4;
```

Strukturo lahko deklariramo kot **typedef**, pri čemer struktura dobi lastni podatkovni tip.

```
typedef struct imeStrukture
{
     podatkovniTip spremenljivka1;
     podatkovniTip spremenljivka2;
     podatkovniTip spremenljivka3;

   }novi_podatkovni_tip;
```

Primer: Deklaracija strukture kot **typedef**

```
#include <stdio.h>

typedef struct person {
    char ime[20];
    char priimek[20];
    int leto;
    float visina;

}ljudje;


int main()
{

    ljudje slovenec;//Novo polje struktur slovenec

    //Vpis v element strukture leto in visina
    slovenec.leto = 1980;
    slovenec.visina = 1.76;

    printf("Leto: %d Visina: %.2f\n", slovenec.leto, slovenec.visina);


}
```

23. Funkcija 'random'

Funkcija generira naključna števila od 0 do neskončno.

```
rand()
```

Če želimo funkcijo random, omejimo na določen interval, zapišemo pa tako:

```
rand()%interval
```

Primer: Funkcija *random*

```
rand()% 10            /*funkcija vrne naključna števila od 0 - 9 */
rand()% 84            /*funkcija vrne naključna števila od 0 - 83*/
(rand()% 84) + 1       /*funkcija vrne naključna števila od 1 - 84*/
```

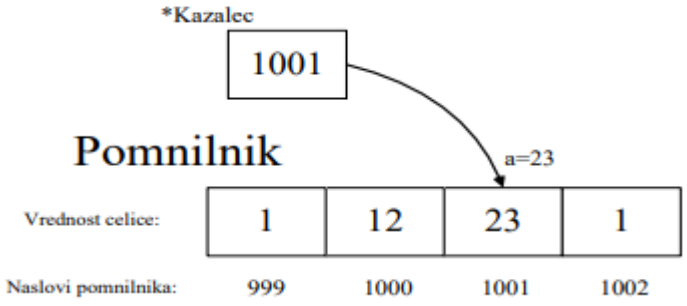
Z uporabo funkcije ***rand*** generiramo enak niz naključnih števil. Če želimo ob vsakem zagonu generirati drug niz naključnih števil, je potrebno uporabiti seme, ki določa razpršenost generatorja. Če spreminjamo seme, se spreminja niz naključnih števil. Seme definiramo s funkcijo ***srand().***

Primer: Uporaba funkcije random s semenom

```
srand(seme)
(rand() % 84) + 1
```

24. Kazalci

Kazalci so spremenljivke, v katerih se shranjujejo naslovi deklariranih spremenljivk (ne vrednosti). Naslov spremenljivke pomeni, da se v kazalec shrani naslov pomnilniške lokacije, na kateri je shranjena vrednost deklarirane spremenljivke.



Slika prikazuje vrednost kazalca **Kazalec*, ki ima vrednost naslova spremenljivke *a*. Če bi imeli vrednost kazalca 1002, bi ta kazal na spremenljivko z vrednostjo 1. Podatkovni tip kazalca je enak kot spremenljivka, za katero uporabimo kazalec.

Deklaracija kazalcev

Kazalec v programskem jeziku C deklariramo s pomočjo * in mu priredimo enak podatkovni tip, kot smo to počeli z ostalimi spremenljivkami. Da dobimo boljši pregled nad spremenljivkami in kazalci, te načeloma poimenujemo z začetno črko k, podčrtajem in številko ali črko (na primer, k_1). S tem pri pregledu kode lažje ločimo med kazalci in spremenljivkami. Poimenovanje kazalca je lahko poljubno.

Primer: `int a, *k_a;`

Pri tej deklaraciji je *a* celoštevilčna spremenljivka (tipa integer), *k_a* je kazalec, ki kaže na spremenljivko tipa integer.

Da kazalcu priredimo naslov spremenljivke *a*, uporabimo simbol &.

Primer: `k_a = &a;`

Sedaj kazalec *k_a* vsebuje naslov spremenljivke *a*. Če želimo odčitati vrednost, na katero kaže kazalec *k_a*, uporabimo simbol *. V tem primeru pomeni, da kazalec preko naslova dostopa do vrednosti spremenljivke *a*.

Z uporabo * lahko odčitamo vrednost spremenljivke, na katero kaže kazalec. **k_a=a*.

Smisel uporabe kazalcev je, da s pomočjo kazalca dostopamo direktno na naslov spremenljivke ali preko naslova operiramo z vrednostjo naslovljene spremenljivke. S kazalci ponavadi ne operiramo enako kot s spremenljivkami (množenje, deljenje …). Nad kazalci najpogosteje uporabljamo inkrement (++) , dekrement (--), seštevanje in odštevanje za pomik po spominski lokaciji.

Primer: Uporaba kazalcev

```c
#include <stdio.h>
main()
{
    int a = 2; // celostevilcna spremenljivka int *k_a; // kazalec na int

    k_a = &a; // kazalec dobi vrednost naslova spremenljivke a

    printf("Vrednost spremenljivke a : %d\n", a);
    printf("Vrednost naslova spremenljivke a: %p\n", &a);
    printf("Vrednost spremenljivke k_a: %p\n", k_a);
    printf("Vrednost kazalca k_a kamor      kaze: %d\n", *k_a);
    printf("Vrednost naslova kazalca k_a : %p\n", &k_a);

    *k_a = 10; // Vpis vrednosti 10 v spremenljivko a

    printf("Nova vrednost spremenljivke a: %d\n", a);
    printf("Vrednost kazalca k_a je naslov spremenljivke a: %p\n ", k_a);

    return 0;
}
```

Izpis programa:

```
Vrednost spremenljivke a : 2
Vrednost naslova spremenljivke a : 0x22FF74
Vrednost spremenljivke k_a : 0x22FF74
Vrednost kazalca k_a kamor kaze : 2
Vrednost naslova kazalca k_a : 0x22FF70

Nova vrednost spremenljivke a : 10
Vrednost kazalca k_a je naslov spremenljivke a : 0x22FF74
```

Kazalec na polje in niz (string)

Kazalec je smiselno uporabljati na poljih in nizih. V tem primeru je potrebno poudariti, da kazalec ne postane polje, ampak še vedno vsebuje samo naslov lokacije, na katero kaže, ter vrednost na tej lokaciji. Kadar kazalec postavimo na polje ali niz, ga lahko postavimo na poljubni element polja ali niza. Kazalec pogosto postavimo na prvi element polja ali niza.

Primer: Deklaracija polja in kazalca

```
int  A[20], * k_A;     // Polje in kazalec
char Niz[120], * k_N;  // Polje in kazalec
```

Primer: Postavitev kazalca na prvi element polja ali niza

```
k_A = &A[0];   //Postavitev na prvi element polja
k_N = &Niz[0]; //Postavitev na prvi element niza
```

Primer: Za prvi element polja ali niza velja krajši zapis.

```
k_A = A;   //Postavitev na prvi element polja, krajši zapis
k_N = Niz; //Postavitev na prvi element niza, krajši zapis
```

Pri tem je potrebno poudariti, da ta zgornji zapis velja le za enodimenzionalna polja. Niz v programskem jeziku je tako lahko samo enodimenzionalen.

Kazalec na enodimenzionalnem polju ali nizu tekom programa uporabljamo enako, kot smo to počeli s poljem ali nizom. Kazalec lahko postavimo na poljubni element polja.

Primer: Postavitev kazalca na poljubni element polja

```
k_A = &A[4];   //Postavitev na peti element polja
k_N = &Niz[2]; //Postavitev na tretji element niza
```

Primer: Izpis enodimenzionalnega polja s kazalcem ter izpis niza

```
#include <stdio.h>

int main()
{
    int polje[4] = { 21,22,23,24 }, * pp; //Polje in kazalec *pp;
    char priimek[20], * pc;               //Niz in kazalec *pc;

   //Kazalca na polje in niz(string)
    pp = &polje[0];    //Kazalec pp na prvi element polja
    pc = &priimek[0];  //Kazalec pc na prvi element niza

   //Krajše-Velja samo za enodimenzionalno polje
    pp = polje;     //Kazalec pp na prvi element polja
    pc = priimek;  //Kazalec pc na prvi element niza


   //Izpis polja brez in s kazalcem
    for (int i = 0; i < 4; i++)
    {
        printf("%d ", polje[i]); //Izpis polja brez in kazalca
    }

    for (int i = 0; i < 4; i++)
    {
        printf("%d ", pp[i]); //Izpis polja s kazalcem
    }


    //Izpis niza brez in s kazalcem
    printf("%s\n", priimek); //Izpis niza brez in kazalca

    for (int i = 0; i < 20; i++)
    {
        printf("%c", pc[i]); //Izpis niza s kazalcem
    }


    //Kazalec na 3. element polja
    pp = &polje[2];

    //Kazalec na 10. element niza
    pc = &priimek[9];


}
```

Kazalec na večdimenzionalno polje postavimo tako, da vpišemo lokacijo celice, na katero želimo postaviti kazalec.

Primer: Postavitev kazalca na večdimenzionalno polje

```
int  P[3][4], * k_P;     // Polje in kazalec
k_P = &P[0][0];          //Postavitev na prvi element polja
```

Primer: Postavitev kazalca na tretjo vrstico in drugi stolpec P

```
k_P = &P[2][1];  //Postavitev na tretjo vrstico in drugi stolpec
```

V tem primeru ne obstaja krajši zapis za postavitev na prvi element polja. Pri večdimenzionalnem polju s kazalcem ne moremo rokovati enako kot s poljem. V tem primeru s kazalcem rokujemo kot z enodimenzionalnim poljem. Lahko si predstavljamo, da se večdimenzionalno polje raztegne v eno dimenzijo. Redosled dimenzije je sledeč: vrstica, stolpec, globina itd. Za dani primer bi se polje P[3][4] obnašalo kot enodimenzionalno polje P[12], *(vrstica* stolpec -> 3*4=12).*

Primer: Polje P[3][4]

	0	1	2	3
0	3	6	8	1
1	56	23	89	9
2	90	100	0	55

Vrednost kazalca k_P na polje P[3][4]

0	1	2	3	4	5	6	7	8	9	10	11
3	6	8	1	56	23	89	9	90	100	0	55

Primer: Kazalec na večdimenzionalno polje

```
#include <stdio.h>

int main()
{
    int polje[3][4] = { {3,6,8,1},{56,23,89,9},{90,100,0,55} }, * k_p;

    //Kazalca na prvi element polje
    k_p = &polje[0][0];

    //Izpis polja brez kazalca
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            printf("%d ", polje[i][j]);
        }
        printf("\n");
```

```
    }
    //Izpis polja s kazalcem - enodimenzionalno
    for (int i = 0; i < 12; i++)
    {
        printf("%d ", k_p[i]);

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

Kazalec lahko uporabimo tudi v podprogramu, v katerem argument funkcije določimo kot kazalec.

Primer: Kazalec, kot argument funkcije

```
#include <stdio.h>

/*Podprogram*/
void podprogram_polje_kazalec(int* Ptr);

/*Glavni program*/
int main()
{
    int polje[3][4] = { {3,6,8,1},{56,23,89,9},{90,100,0,55} }, * k_p;

    //Kazalca na prvi element polje
    k_p = &polje[0][0];

    //Klic funkcije za izpis preko kazalca
    podprogram_polje_kazalec(k_p);


    //Izpis polja brez kazalca
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            printf("%d ", polje[i][j]);
        }
        printf("\n");
    }



    printf("\n");

}

/*Podprogram */
void podprogram_polje_kazalec(int* Ptr)
{
    //Izpis polja s kazalcem - enodimenzionalno
    for (int i = 0; i < 12; i++)
    {
        printf("%d ", Ptr[i]);

    }
}
```

Kazalec na strukturo

Kazalec na strukturo postavimo enako kot na poljubno polje ali niz, tako da določimo element. Za dostop element v strukturi preko kazalca uporabimo namesto pike operator '->'.

Primer: Kazalec na strukturo ter uporaba podprograma

```c
#include <stdio.h>


/*Struktura*/
typedef struct person {
    char ime[20];
    char priimek[20];
    int leto;
    float visina;

}ljudje;

/*Podprogram*/
void podprogram_struktura_kazalec(int* Ptr);

/*Glavni program*/
int main()
{

    ljudje slovenec[20], * strPtr;//Novo polje struktur slovenec in kazalec

    //Postavitev kazalca na drugi element polja slovenec
    strPtr = &slovenec[1];


    //Vpis v element strukture leto in visina preko kazalca strPtr
    strPtr->leto = 1980;
    strPtr->visina = 1.76;

    //Izpis brez kazalca
    printf("Leto: %d Visina: %.2f\n", slovenec[1].leto, slovenec[1].visina);

    //Izpis s kazalcem
    printf("Leto: %d Visina: %.2f\n", strPtr->leto, strPtr->visina);

    //Klic podprograma
    podprogram_struktura_kazalec(strPtr);
}

/*Podprogram*/
void podprogram_struktura_kazalec(int* Ptr)
{
    //Izpis s kazalcem
    printf("Leto: %d Visina: %.2f\n", Ptr->leto, Ptr->visina);
}
```

Kazalec na podprogram ali funkcijo

Kazalec lahko prav tako uporabimo za podprogram, kjer kazalec ne kaže več na lokacijo spremenljivke, ampak na lokacijo v pomnilniku, kjer se nahaja podprogram. Deklaracija kazalca mora biti enaka kot deklaracija funkcije ter argumenti.

```
Function_declaration (*Pointer_to_fun)(function_arguments) = &function_name;
```

Primer: Preprost primer kazalca na funkcijo

```c
#include <stdio.h>

/*Podprogram*/
void fun_out(int st)
{
    printf("Izpis vhoda: %d\n", st);
}

//Glavni program
int main()
{
    // Kazalec fun_Kazalec na funkcjio fun_out
    void (*fun_Kazalec)(int) = &fun_out;


    // Uporaba kazalca za klic funkcije fun_out
    (*fun_Kazalec)(7);


}
```

Funkcija je deklarirana kot *void* in ima en argument *int* . Kazalec je zato deklariran kot void in ima en argument int. `void (*fun_Ptr)(int) = &fun_out;`

Primer: Primer, ko funkcija ni prazna in ima dva različna argumenta.

```c
#include <stdio.h>

/*Podprogram*/
float fun_out(int st1, float st2)
{
    printf("Izpis vhoda: %d %.3f\n", st1, st2);
    return st1 * st2;
}

/*Glavni program*/
int main()
{
    // Kazalec fun_Ptr na funkcjio fun_out
    float num;
    float (*fun_Ptr)(int, float) = &fun_out;


    // Uporaba kazalca za klic funkcije fun_out z dvema argumentoma
    num = (*fun_Ptr)(7, 2.5);
    printf("%f \n", num);

}
```

V tem primeru je kazalec deklariran kot float, enako kot podprogram. Funkcija ima dva argumenta int in float. `float (*fun_Ptr)(int,float) = &fun_out;`

Primer: Program, kjer je argument prve funkcije druga funkcija. To je možno narediti preko uporabe kazalca.

```
#include <stdio.h>

/*Podprogram 1*/
void fun_1(int st1, float st2)
{
    printf("Izpis vhoda fun_1: %d %.3f\n", st1, st2);

}

/*Podprogram 2*/
int fun_2(int st3, void (*funPtr)(int, float))
{
    //Klic funkcije fun_1 preko kazalca funPtr
    funPtr(7, 2.5);
    printf("Izpis vhoda fun_2 : % d\n", st3);
    return st3 * 3;
}

/*Glavni program*/
int main()
{
    // Kazalec fun_Ptr na funkcjio fun_out
    int num;

    // Klic funkcije, kjer je drugi argument funkcija fun_1
    num = fun_2(2, fun_1);
    printf(" % d \n", num);

}
```

V tem primeru je drugi argument funkcije *fun_2* kazalec na funkcijo tipa void z dvema argumentoma tipa *int* in *float*. `int fun_2(int st3, void (*fun)(int, float)`. Klic funkcije ter prenos funkcije *fun_1* v argument *fun_2* je preprost, kjer v argument funkcije *fun_2* vstavimo ime funkcije *fun_1*, `num = fun_2(2, fun_1)`. Klic funkcije *fun_1* v *fun_2* se izvede preko kazalca s pripadajočimi argumetni funkcije *fun_1*, `funPtr(7, 2.5)`.

Primer: Program, kjer kazalec postavimo na več funkcij, ki imajo enako deklarirane argumente.

```c
#include <stdio.h>


/*Podprogram fun_1*/
void fun_1(int a, float b)
{
    printf("Izpis vhoda fun_1: %d %.3f\n", a, b);
}

/*Podprogram fun_2*/
void fun_2(int c, float d)
{
    printf("Izpis vhoda fun_2: %d %.3f\n", c, d);
}

/*Podprogram fun_3*/
void fun_3(int e, float f)
{
    printf("Izpis vhoda fun_3: %d %.3f\n", e, f);
}

//Glavni program
int main()
{
    // Kazalec fun_Ptr na funkcjie
    void (*fun_Ptr[])(int, float) = { fun_1, fun_2, fun_3 };

    // Klic funkcije fun_1 preko kazalca *fun_Ptr[0]
    (*fun_Ptr[0])(0, 2.3);
    // Klic funkcije fun_2 preko kazalca *fun_Ptr[1]
    (*fun_Ptr[1])(1, 4.1);
    // Klic funkcije fun_3 preko kazalca *fun_Ptr[2]
    (*fun_Ptr[2])(2, 0.3);

}
```

25. Standardni vhodi/izhodi

Datoteke uporabljamo za trajnejše shranjevanje podatkov. Na primer, ko se program konča ali se računalniški sistem ugasne, kreirana datoteka lahko vsebuje podatke in informacije o delovanju programa. Prav tako je podatke iz datoteke možno prenašati med različnimi programi in računalniškimi sistemi. Datoteke lahko upravljamo na različne načine: jih ustvarimo, brišemo, zapremo, beremo in pišemo. V programskem jeziku lahko ustvarimo tekstovno *'.txt'* in binearno '***.bin***' datoteko. Tekstovna datoteka vsebuje znake, ki jih predpisuje ASCII-tabela. S takšno vrsto datoteke je preprosto upravljati, zavzame več pomnilniškega prostora in je manj varna. Binearna datoteka vsebuje samo zaporedje ničel in enic, tako je možno shraniti več podatkov na enakem pomnilniškem prostoru, je varnejša, toda rokovanje z datoteko je zahtevnejše.

Operacije z datotekami:

- Kreiranje nove datoteke
- Odpiranje obstoječe datoteke
- Zapiranje datoteke
- Branje in pisanje v datoteko

Glede na operacije z datotekami uporabljamo različne programske ukaze. Preden se lotimo operacije z datotekami, je na začetku programa potrebno deklarirati kazalec na datotečno strukturo, ki se nahaja v knjižnici *'stdio.h'*.

```
FILE *fPtr;
```

Kjer je ***fPtr*** uporabniško kreiran kazalec na datotečni sistem.

Kreiranje datoteke

Za kreiranje datoteke uporabljamo ukaz ***fopen().***

```
fPtr = fopen("ime_datoteke", "način");
```

Kjer ime datoteke ustvarimo kot ***.txt*** ali ***.bin*** datoteko. Način, kako bomo uporabljali datoteko, je podan v spodnji tabeli.

Po koncu uporabe datoteke jo zapremo z ukazom.

```
fclose(fPtr);
```

Različni načini za upravljanje z datotekami [7]:

Tabela 20: Načini za upravljanje z datotekami

Način	Pomen	Med izvajanjem
r	Odpiranje za branje (read)	Če datoteka ne obstaja, funkcija ***fopne()*** vrne ***NULL.***
rb	Odpiranje za branje binearne datoteka (read binary)	Če datoteka ne obstaj,a funkcija ***fopne()*** vrne ***NULL.***
w	Odpiranje za pisanje v datoteko (write)	Če datoteka obstaja, se vsebina datoteke izbriše in se omogoči ponovno vpisovanje v prazno datoteko. Če datoteka še ne obstaja, se kreira na novo.
wb	Odpiranje za pisanje v binearno datoteko (write binary)	Če datoteka obstaja, se vsebina datoteke izbriše in se omogoči ponovno vpisovanje v prazno datoteko. Če datoteka še ne obstaja, se kreira na novo.
a	Odpiranje za dodajanje podatkov na konec datoteke (append)	Če datoteka ne obstaja, se kreira na novo. Prejšnja vsebina datoteke se ne briše.
ab	Odpiranje za dodajanje podatkov na konec binearne datoteke (append binary)	Če datoteka ne obstaja, se kreira na novo. Prejšnja vsebina datoteke se ne briše.
r+	Odpiranje za branje in pisanje	Če datoteka ne obstaja, funkcija ***fopne()*** vrne ***NULL.***
rb+	Odpiranje za branje in pisanje v binearno datoteko	Če datoteka ne obstaja, funkcija ***fopne()*** vrne ***NULL.***
w+	Odpiranje za branje in pisanje	Če datoteka obstaja, se vsebina datoteke izbriše in se omogoči ponovno vpisovanje v prazno datoteko. Če datoteka še ne obstaja, se kreira na novo.
wb+	Odpiranje za branje in pisanje v binearno datoteko	Če datoteka obstaja, se vsebina datoteke izbriše in se omogoči ponovno vpisovanje v prazno datoteko. Če datoteka še ne obstaja, se kreira na novo.
a+	Odpiranje za branje in dodajanje	Če datoteka ne obstaja, se kreira na novo.
ab+	Odpiranje za branje in dodajanje v binearno datoteko	Če datoteka ne obstaja, se kreira na novo.

Funkcije za vpis in izpis v datoteko

Funkcije za operiranje z datotekami se nahajajo v knjižnici *'stdio.h'*. Predstavljamo nekaj najpomembnejših.

Funkcije za splošno rokovanje z datotekami

- **Odpiranje datoteke – fopen**

```
FILE *fopen(const char *filename, const char *mode)
```
Primer: `fPtr = fopen("My_file.txt", "a");`

- **Zapiranje datoteke – fclose**

```
int fclose(FILE *fPtr)
```
Primer: `fclose(fPtr);`

- **Na začetek datoteke – rewind**

```
void rewind(FILE *fPtr)
```
Primer: `rewind (fPtr);`

- **Brisanje datoteke – remove**

```
int remove(const char *filename)
```
Primer: `remove("My_file.txt");`

- **Preimenovanje obstoječe datoteke – rename**

```
int rename(const char *filename_old, const char *filename_new)
```
Primer: `rename ("My_file.txt","My_file_new.txt");`

Funkcije za branje

- **Branje niza iz datoteke – fread**

```
size_t fread(char *buffer, size_t len, size_t byte_num, FILE *fPtr)
```
Primer: `fread(line_data, 10 , 1 , fPtr);`

Branje 10 znakov, kjer je znak dolžine 1 bajt.

- **Branje karakterja iz datoteke – fgetc**

```
int fgetc(FILE *fPtr)
```
Primer: `char c = fgetc(fPtr);`

- **Branje niza (stringa) iz datoteke – fgets**

```
char *fgets(char *buffer, int rec_len, FILE *fPtr)
```
Primer: `fgets(line_data, sizeof (line_data), fPtr);`

- **Branje niza iz datoteke s formatiranjem – fscanf**

```
int fscanf(FILE *fPtr, const char *format, ...)
```
Primer: `fscanf(fPtr,"%s %s %s", str1, str2, str3);`

Funkcije za pisanje

- **Pisanje niza iz datoteke – fwrite**

```
size_t fwrite(char *buffer, size_t len, size_t byte_num, FILE
    *fPtr)
```

Primer: `fwrite(line_data, 10 , 1 , fPtr);`

Pisanje 10 znakov, kjer je znak dolžine 1 bajt.

- **Pisanje niza v datoteko s formatiranjem – fprintf**

```
char *fprintf(FILE *fPtr, "format data", data,data1,…)
```

Primer: `fprintf(fptr,"%d", num);`

- **Zapisovanje enega znaka v datoteko – fputc**

```
int fputc(int char, FILE *fPtr)
```

Primer: `fputc("A", fPtr);`

- **Zapisovanje niza v datoteko – fputs**

```
int fputs(int char *str, FILE *fPtr)
```

Primer: `fputs("Hello!",fPtr);`

Primer: Kreiranje datoteke ter vpis poljubnega besedila z ukazom ***fgets()***, *'fgets (get-string)'*

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

#define DATA_SIZE 1000

int main()
{
    /* Deklaracija spremenljivk */
    char data[DATA_SIZE];
    FILE* fPtr;

    /* Kreiranje txt datoteke v pisalnem nacinu "w" */
    fPtr = fopen("nova_datoteka.txt", "w");


    /* Vpis besedila v string */
    printf("Vsebina za datoteko: \n");
    fgets(data, DATA_SIZE, stdin);

    /* Vpis stringa v datoteko */
    fputs(data, fPtr);

    /* Zapiranje datoteke */
    fclose(fPtr);

}
```

Namesto funkcij ***fgets()***, lahko uporabimo tudi ***scanf_s****("%s", &data, DATA_SIZE).*
Funkcijo ***fputs*** *()* lahko nadomestimo z funkcijo ***fprintf****(fPtr, "%s", data).*

Primer: Dodajanje števila v obstoječo datoteko, ki ni prazna

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

#define DATA_SIZE 1000

int main()
{
    /* Deklaracija spremenljivk */
    char data[DATA_SIZE];
    FILE* fPtr;
    float num;

    /* Odpiranje/Kreiranje txt datoteke v append nacinu “a” */
    fPtr = fopen("append_number.txt", "a");

    /* Vpis stevila */
    printf("Add number to file: \n");
    scanf_s("%f", &num);

    /*Pretvorba stevila v string*/
    sprintf(data, "%.2f\n", num);

    /* Vpis stringa v datoteko */
    fputs(data, fPtr);

    /* Zapiranje datoteke */
    fclose(fPtr);
}
```

Primer: Štetje vrstic v datoteki in izpis poljubne vrstice

Glavni program

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

#define DATA_SIZE 1000
int line_number(char* file_name);
void read_line(char* file_name, int line, char* out_line_string);

int main()
{
    /* Deklaracija spremenljivk */
    char file_name[] = "Line_number.txt", * FileNamePtr;
    char Line_string[DATA_SIZE], * DataPtr;
    int lines;            //MAX. stevilo vrstic v datoteki
    int lin_read_num = 1;   //Branje iz poljubne vrstice

    /*Postavitev kazalca na string*/
    DataPtr = Line_string;
    FileNamePtr = file_name;

    /*Klic funkcije stevilo vrstic*/
    lines = line_number(FileNamePtr);
    read_line(FileNamePtr, lin_read_num, DataPtr);

    /*Izpis*/
    printf("Stevilo vrstic: %d.\n", lines);
    printf("Vrstica %d. Line text: %s\n ", lin_read_num, Line_string);
}
```

Podprogram za štetje vrstic v datoteki

```
int line_number(char* file_name)
{
    int lines_num = 1;
    char znak;
    FILE* fPtr;

    /*Odpiranje datoteke v bralnem nacinu “r” */
    fPtr = fopen(file_name, "r");

    /*Sporocilo ce datoteka ne obstaja*/
    if (fPtr == NULL)
    {
        printf("File doesn't exist!\n");
        return 0;
    }

    /*Branje vsakega znaka v datoteki in iskanje nove vrstice '\n' dokler
     'EOF' – end of file*/
    for (znak = getc(fPtr); znak != EOF; znak = getc(fPtr))
    {
        if (znak == '\n') //Pvecaj stevec ce je nova vrstica
        {
            lines_num++;
        }
    }
    /* Zapiranje datoteke */
    fclose(fPtr);
    return lines_num;
}
```

Podprogram za izpis poljubne vrstice v datoteki

```
void read_line(char* file_name, int line, char* out_line_string)
{
    int lines_num = 0;
    char znak;
    FILE* fPtr;
    char line_data[1000];
    char* current_line_data;

    /*Odpiranje datoteke v bralnem nacinu “r” */
    fPtr = fopen(file_name, "r");

    /*Sporocilo ce datoteka ne obstaja*/
    if (fPtr == NULL)
    {
        printf("File doesn't exist!\n");
    }


    //Branje posamezne vrstice
    while (fgets(line_data, sizeof(line_data), fPtr))
    {
        lines_num++; //Stetje vrstic
        if (lines_num == line)
        {
            //kopiraj vsebino stringa na kazalec out_line_string
            memcpy(out_line_string, line_data, sizeof(line_data));

        }
    }

    /* Zapiranje datoteke */
    fclose(fPtr);

}
```

26. ASCII-tabela

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

Terminološko kazalo:

T

U

V

W

Z

Reference:

1. Mark Lutz, *Learning Python 5ed: Powerful Object-Oriented Programming*, O'Reilly, 2013.
2. Philip Robbins, *Python Programming for Beginners: The Complete Guide to Mastering Python in 7 Days with Hands-On Exercises – Top Secret Coding Tips to Get an Unfair Advantage and Land Your Dream Job!*, Independently published, 2023.
3. David Beazley, *Python Distilled (Developer's Library)*, Pearson ,2021.
4. Johannes Ernesti, Peter Kaiser, Python 3: The Comprehensive Guide, Rheinwerk Computing, 2022.
5. Greg Perry, *C Programming Absolute Beginner's Guide*, Que Publishing, 2013.
6. David Griffiths, Head First C: A Brain-Friendly Guide, Shroff , 2012.
7. Mike McGrath, *C Programming in easy steps, 5th Edition*, In Easy Steps Limited, 2018.