

IO operacije GNU

Ukoliko želimo da nam program radi bilo šta korisno, neophodno je obezbediti ulaz/izlaz podataka, bez obzira na to da li se oni unose sa tastature i prikazuju na monitoru, ili se u tu svrhu koristi hard-disk računara. Dok C programski jezik ne obezbeđuje puno opcija u ovom smeru, GNU C biblioteka sadrži brojne funkcije, od kojih ćemo u ovom poglavlju pokriti samo one osnovne.

Svi objekti od kojih možemo očekivati ulazne podatke, kao i oni kojima možemo prosleđivati izlazne u GNU sistemu se tretiraju kao datoteke (fajlovi): ne samo datoteke sa hard diska (objektni fajlovi, C izvorni fajlovi, ASCII tekstualni fajlovi, izvršni fajlovi), već i periferije kao što su štampač, tastatura, monitor, ... Kada se piše C program koji korisniku treba da omogući unos sa tastature, program tada čita iz, ili prihvata ulaz od strane tastature, na gotovo identičan način na koji bi čitao podatke iz neke datoteke. Slično, kada C program prikazuje tekst na korisničkom monitoru, izlazni podaci se šalju terminalu, na identičan način kao što bi se tekstualni string upisivao u tekstualni fajl. Zapravo, u mnogim slučajevima ćemo čak i koristiti iste funkcije kako bismo pročitali tekst sa tastature ili iz tekstualnog fajla, odnosno slali podatke na terminal, kao što bismo upisivali podatke u tekstualni fajl.

Dakle, C tretira periferije u mikroracunarskom sistemu kao datoteke, poznate kao datoteke uređaja (device files), pri čemu svaki od njih ima svoje ime: /dev/lp0 (device, line printer 0), /dev/fd0 (device floppy drive 0)... Prednost ovakvog pristupa leži u činjenici da nije neophodno poznavati kako neki uređaj radi, jedino nam je značajan interfejs preko koga možemo da mu pristupamo i komuniciramo sa njim. Na primer, C program često prihvata podatke kao ulaz sa tastature, kojoj se C obraća preko naziva datoteke stdin (skraćeno od "standard input"), dok se izlaz često šalje na monitor koji je u sistemu prepoznat kao stdout. Često se ulaz/izlaz može preusmeriti na drugu datoteku (npr tekstualni fajl), korišćenjem standardne Linux direktive za preusmeravanje (>,<).

Pre nego što se pročita sadržaj datoteke, ili se u nju upisuje nova vrednost, neophodno je otvoriti datoteku, obično korišćenjem fopen komande koja vraća *tok* (eng. *stream*) ili korišćenjem open komande koja vraća *deskriptor datoteke* (*file descriptor*). Datoteka se može otvoriti za čitanje, pisanje ili oba. Takođe, moguće je otvoriti datoteku tako što se novi sadržaj dodaje na prethodni sadržaj datoteke (*append mod*).

Osim u retkim slučajevima, datotekama se pristupa iz ostalih C funkcija ne preko njihovog imena, već preko tokova, odnosno deskriptora datoteka:

```
//upis u datoteku, kao i zatvaranje datoteke vrši se korišćenjem toka my_stream
fprintf (my_stream, "Just a little hello from fprintf.\n");
close_error = fclose (my_stream);
```

Sa druge strane, fopen funkcija prima ime datoteke, i vraća tok, koji se kasnije koristi za pristup datoteci:

```
my_stream = fopen (my_filename, "w");
```

Na ovaj način se mapira naziv datoteke u tok ili deskriptor datoteke koji se kasnije koristi, naziv datoteke se koristi samo tokom otvaranja datoteke, dok se kasnije koriste samo tokovi ili deskriptori datoteka.

Datotekama se može pristupati na *visokom nivou* (high level) ili *niskom nivou* (low level). Pristup datoteci na visokom nivou znači da se datoteka koristi na višem nivou apstrakcije i ovo obezbeđuje obično sigurniji način pristupa koji je ujedno i komotniji i elegantniji u poređenju sa pristupom na niskom nivou. Stoga ćemo se ovde bazirati uglavnom na ovom pristupu datotekama na visokom nivou, a samo ćemo pomenuti interfejs za pristup datotekama na niskom nivou.

Veza sa datotekom otvorenom na visokom nivou se naziva tok, dok se u slučaju konekcije na niskom nivou, ona naziva deskriptor datoteke. Ove dve veze imaju različite tipove, kao što ćemo videti kasnije. Većini funkcija koje su zadužene za rad sa ulaznim/izlaznim podacima se prosleđuje ili tok, ili deskriptor datoteke.

Interesantno je da, iako se tokovi smatraju za vezu na visokom nivou, a deskriptori datoteka na niskom nivou, GNU sistem podržava oba pristupa, dok većina UNIX-like sistema preferira tokove.

Nakon što se završi sa korišćenjem datoteke, neophodno je zatvoriti datoteku, nakon čega više nije moguće koristiti je ponovo (do sledećeg otvaranja).

1. Rutine za rad sa datotekama na visokom nivou

Ove rutine se obično lako prepoznaju po slovu "f" na početku naziva funkcije: otvaranje datoteke se vrši funkcijom *fopen*, dok se u slučaju pristupa na niskom nivou ova funkcija zove *open*. Neke od ovih funkcija imaju naziv sličan funkcijama sa kojima ste već familijarni, na primer funkcija *fprintf* se ponaša isto kao i *printf* funkcija, sa razlikom što se umesto ispisa na monitor, podaci formatirano upisuju u datoteku.

1.1. Otvaranje datoteke

Osnovna funkcija za rad sa datotekama na visokom nivou je *fopen*. Kada se otvori datoteka, GNU C biblioteka kreira novi tok i konekciju sa datotekom. Ukoliko se ovoj funkciji prosledi naziv datoteke koja ne postoji, ona će biti kreirana. *Fopen* funkcija vraća tok koji je kreiran prilikom uspostavljanja konekcije sa datotekom.

Tok je predstavljen promenljivom tipa `FILE*`, dok *fopen* vraća null pointer ukoliko nije uspešno otvorena datoteka.

Prvi parametar funkciji je string koji sadrži naziv datoteke koja se otvara. String može i ne mora biti const promenljiva:

```
FILE *my_stream;  
my_stream = fopen ("foo", "r");
```

```
FILE *my_stream;  
char *my_filename = "foo";  
my_stream2 = fopen (my_filename, "r");
```

Drugo parametar je takođe string koji sadrži jedan od sledećih karaktera:

r	Otvori datoteku samo za čitanje. Datoteka mora da postoji.
w	Otvori datoteku samo za upisivanje. Ako datoteka postoji, njen prethodni sadržaj će biti obrisan, a ukoliko ne postoji, biće kreirana.
r+	Otvori datoteku za čitanje i upis. Datoteka mora da postoji. Sadržaj datoteke se inicijalno ne menja, ali je pozicija unutar datoteke postavljena inicijalno na početak.
w+	Otvori datoteku za čitanje i upis. Ukoliko datoteka postoji, njen prethodni sadržaj će biti obrisan, a ukoliko ne postoji, datoteka će biti kreirana.
a	Otvori datoteku samo za dodavanje sadržaja, što je isto kao i upis u datoteku, sa razlikom da se novi podaci upisuju samo na kraj već postojeće datoteke. Ako datoteka ne postoji, biće kreirana.
a+	Otvori datoteku samo dodavanje sadržaja i čitanje. Ako datoteka postoji, njen sadržaj se neće menjati sve dok se ne doda novi sadržaj, a ukoliko ne postoji, biće kreiran. Inicijalna pozicija za čitanje unutar datoteke je na početku datoteke, ali je pozicija za dodavanje na kraju datoteke.

Moguće je dodati karakter *x* nakon svakog karaktera iz tabele navedene iznad. Ovaj karakter podrazumeva neuspešno korišćenje *fopen* funkcije ukoliko datoteka već postoji i na ovaj način se štitimo od slučajnog brisanja datoteke prilikom otvaranja datoteke.

Sledeći primer prikazuje korektno korišćenje *fopen* funkcije za otvaranje tekstualne datoteke koja se čita. Nakon početnog pokretanja programa, pokušajte ponovo pozvati program nakon što datoteka već postoji (u Linux-u se ovo najlakše radi sa *touch snazzyjazz.txt*)

```

#include <stdio.h>

int main()
{
    FILE *my_stream;

    my_stream = fopen ("snazzyjazz.txt", "r");

    if (my_stream == NULL)
    {
        printf ("File could not be opened\n");
    }
    else
    {
        printf ("File opened! Closing it now...\n");
        /* Close stream; skip error-checking for brevity of example */
        fclose (my_stream);
    }
    return 0;
}

```

1.2. Zatvaranje datoteke

Funkcija koja se koristi u ovu svrhu je *fclose*, kojoj se prosleđuje tok a ona će prekinuti vezu sa odgovarajućom datotekom, pre čega će pročitati bilo koji baferovani ulaz, ili upisati baferovani izlaz (biće kasnije više reči o baferovanim ulazima/izlazima). Ukoliko je datoteka uspešno zatvorena, vraća se vrednost 0, u suprotnom vraćena vrednost je EOF.

Važno je proveriti eventualnu grešku prilikom zatvaranja toka koji se koristio za upis. Na primer, nakon što *fclose* pokuša da upiše u datoteku preostale podatke u baferu, moguće je da se generiše greška jer na disku nema dovoljno prostora.

```

#include <stdio.h>

int main()
{
    FILE *my_stream;
    char my_filename[] = "snazzyjazz.txt";
    int close_error;

    my_stream = fopen (my_filename, "w");
    fprintf (my_stream, "Just a little hello from fprintf.\n");

    close_error = fclose (my_stream);

    if (close_error != 0)
    {
        printf ("File could not be closed.\n");
    }
    else
    {
        printf ("File closed.\n");
    }

    return 0;
}

```

1.3. Otvaranje i zatvaranje tokova u C++

C++ obezbeđuje posebne klase za upis i čitanje iz datoteka:

`ofstream`: Klasa za rad sa tokom za upis u datoteku

`ifstream`: Klasa za rad sa tokom za čitanje iz datoteke

`fstream`: Klasa koja se može koristiti i za upis iz čitanje iz datoteke

Ove tri klase nasleđuju direktno ili indirektno od klasa `istream` i `ostream`. Već ste navikli, ako ste iole koristili C++ na rad sa objektima ovih klasa: `cin` je objekat klase `istream`, dok je `cout` objekat klase `ostream`. Tokove navedene iznad korišćemo na isti način kao što smo koristili i `cin/cout`, sa jedinom razlikom što ove tokove moramo asociirati sa odgovarajućim datotekama. Pogledajmo primer:

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    ofstream myfile;
    myfile.open ("example.txt");
    myfile << "Writing this to a file.\n";
    myfile.close();
    return 0;
}
```

Kao rezultat programa iznad, kreiraće se datoteka sa nazivom `example.txt` i u nju će biti upisana data rečenica na isti način kao što smo npr. koristili `cout` da prikazemo tekst na monitoru.

Prva operacija koju vršimo na toku jeste da asociramo taj tok sa određenom datotekom. Ova operacija je poznata kao otvaranje datoteke. Otvorena datoteka, baš kao u programskom jeziku C, u okviru programa je predstavljena kao tok, tj. objekat jedne od gore navedenih klasa. Bilo koji unos ili čitanje primenjeni na toku, biće primenjeni na datoteku kojoj je dati tok pridružen.

Da bismo otvorili datoteku korišćenjem toka, koristimo metodu klase `open`:

```
open (filename, mode);
```

Gde `filename` predstavlja string koji reprezentuje naziv datoteke koja se otvara a `mode` je opcioni parameter koji je kombinacija sledećih indikatora (flegova):

<code>ios::in</code>	Otvora se za upis
<code>ios::out</code>	Otvora se za čitanje
<code>ios::binary</code>	Otvora se u binarnom modu
<code>ios::ate</code>	Postavlja inicijalni pokazivač unutar datoteke na kraj. Ako nije postavljen ovaj fleg, inicijalna pozicija će biti na početku datoteke
<code>ios::app</code>	Sav upis u datoteka će se izvršavati na kraju datoteke, dodajući ga već postojećem sadržaju (<i>append</i>)
<code>ios::trunc</code>	Ako je već postojeća datoteka otvorena za upis, prethodni sadržaj će biti obrisani i zamenjeni novim

Svi ovi indikatori mogu se kombinovati korišćenjem operatora `|`. Na primer, ako želimo da otvorimo datoteku `example.bin` u binarnom modu i dodamo podatke na već postojeće to možemo učiniti na sledeći način::

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Svaka metoda `open` u bilo kojoj od klasa `ofstream`, `ifstream` i `fstream` ima već podrazumevane indikatore podešene u slučaju da se datoteka otvara bez drugog argumenta u metodi `open`:

Klasa Podrazumevani mod

```
ofstream ios::out
ifstream ios::in
fstream ios::in | ios::out
```

Kod `ifstream` i `ofstream` klasa, `ios::in` i `ios::out` su automatski podešeni i podrazumevani, čak i ako ih mod koji je postavljen kroz drugi argument ne uključuje.

Za `fstream`, podrazumevana vrednost je primenjena jedino ako je `open` metoda pozvana bez drugog argumenta. Ako je ona pozvana sa bilo kojom kombinacijom indikatora, podrazumevan mod će biti odbačen i korišćen umesto njega mod definisan tim drugim argumentum metode `open` (neće biti kombinovanja kao u slučaju klasa `ifstream` i `ofstream`).

Tokovi otvoreni u binarnom modu operacije vrše upise i čitanja nezavisno od formatiranja. Datoteke koje nisu otvorene u binarnom modu su poznate kao tekstualne datoteke i kod njih može doći do određenih preformatiranja podataka koji se čitaju ili upisuju (kao naprimer `\n` za novi red i slično).

Obzirom na to da je najčešće otvaranje datoteke prva akcija koja se vrši na njoj, sve tri gore navedene klase imaju konstruktor koji automatski poziva `open` metodu i prihvata iste argumente kao i ona. Dakle, možemo da uradimo sledeće:

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

kombinujući kreiranje objekta i otvaranje toka u jednoj naredbi. Oba oblika za otvaranje datoteke su validna i ekvivalentna.

Da bismo proverili da li je tok uspešno otvoren, to možemo da uradimo pozivanjem metode `is_open`. Ova metoda vraća `bool` vrednost `true` u slučaju da je tok pridružen otvorenoj datoteci, odnosno `false` u suprotnom:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

Kada smo završili sa upisom ili čitanjem u datoteku, potrebno je da je zatvorimo kako bismo obavestili operativni sistem o tome i kako bi resursi postali ponovo dostupni. U tom cilju koristimo metodu `close` na toku. Ova metoda prebacuje sve podatke iz toka u datoteku (ako to nije već ranije urađeno) i zatvara datoteku:

```
myfile.close();
```

Kada je pozvana ova metoda, tok objekat može ponovo da se koristi sa novom datotekom, a i sama datoteka prethodno korišćena postaje slobodna za ponovno korišćenje od strane drugih procesa.

U slučaju da je objekat uništen dok je jos uvek asociiran sa nekom datotekom, destructor će automatski pozvati metodu `close`.

1.4. Ulaz/izlaz u blokovima

Dve funkcije koje se koriste u ovu svrhu su `fread` i `fwrite` i one omogućavaju upis i čitanje blokova fiksne veličine, nasuprot upisu/čitanju red po red, ili karakter po karakter. Ove dve funkcije takođe mogu da se koriste u cilju upisa i čitanja blokova binarnih podataka. Ova karakteristika je vrlo interesantna kada je potrebno čuvanje podataka u istom formatu kako ih koristi naš program: na primer, moguće je smeštanje kompletnog multidimenzionog niza floating-point vrednosti u datoteku korišćenjem `fwrite` funkcije, a onda ga pročitati direktno nazad korišćenjem `fread` funkcije, bez ikakvog gubitka informacija koje bi se desile da smo na neki način konvertovali ove brojeve u tekst (stringove), npr funkcijom `fprintf`. Osnovni nedostatak ovakvog pristupa, nasuprot korišćenja formatiranih ASCII tekstualnih datoteka, leži u činjenici da ovakve datoteke nije jednostavno čitati i editovati korišćenjem tekst editora).

Kako bi se upisao niz `my_array`, koji sadrži `object_count` objekata, pri čemu je svaki veličine `object_size`, korišćenjem toka `my_stream`, koristi se:

```
fwrite (&my_array, object_size, object_count, my_stream);
```

Prilikom čitanja:

```
fread (&my_array, object_size, object_count, my_stream);
```

`Fwrite` funkcija prima četiri parametra: prvi parametar je void pokazivač (`void *`) (**ZAŠTO???**) na niz koji sadrži podatke koji će se upisivati u datoteku. Drugi parametar je promenljiva tipa `size_t` i odnosi se na veličinu objekta koji se upisuje, dok je treći parametar, takođe tipa `size_t` i specificira koliko takvih objekata upisujemo. Poslednji parametar je tok koji se koristi za upis i mora biti tipa `FILE*`. Ukoliko vraćena vrednost funkcije `fwrite` nije jednaka trećem parametru, desila se neka greška prilikom upisa.

Slično kao `fwrite`, `fread` funkcija prima četiri parametra koji su potpuno ekvivalentni. Opet, povratna vrednost mora biti jednaka vrednosti trećeg argumenta funkcije, u suprotnom se desila neka greška.

U sledećem primeru, kreira se niz koji je popunjen umnošcima broja 2, zatim je ispisan na monitoru, upisan u datoteku, nakon čega je izmenjen i ponovo ispisan. Na kraju, nakon ponovnog učitavanja iz datoteke, pokazuje se da je sadržaj isti kao pre izmena.

```
#include <stdio.h>

int main()
{
    int row, column;
    FILE *my_stream;
    int close_error;
    char my_filename[] = "my_numbers.dat";
    size_t object_size = sizeof(int);
    size_t object_count = 25;
    size_t op_return;

    int my_array[5][5] =
    {
        2,  4,  6,  8, 10,
        12, 14, 16, 18, 20,
        22, 24, 26, 28, 30,
        32, 34, 36, 38, 40,
        42, 44, 46, 48, 50
    };
    printf ("Initial values of array:\n");
    for (row = 0; row <= 4; row++)
    {
        for (column = 0; column <=4; column++)
        {
            printf ("%d  ", my_array[row][column]);
        }
    }
}
```

```

        printf ("\n");
    }

my_stream = fopen (my_filename, "w");
op_return = fwrite (&my_array, object_size, object_count, my_stream);
if (op_return != object_count)
    {
        printf ("Error writing data to file.\n");
    }
else
    {
        printf ("Successfully wrote data to file.\n");
    }

/* Close stream; skip error-checking for brevity of example */
fclose (my_stream);

printf ("Zeroing array...\n");
for (row = 0; row <= 4; row++)
    {
        for (column = 0; column <=4; column++)
            {
                my_array[row][column] = 0;
                printf ("%d ", my_array[row][column]);
            }
        printf ("\n");
    }

printf ("Now reading data back in...\n");
my_stream = fopen (my_filename, "r");
op_return = fread (&my_array, object_size, object_count, my_stream);
if (op_return != object_count)
    {
        printf ("Error reading data from file.\n");
    }
else
    {
        printf ("Successfully read data from file.\n");
    }
for (row = 0; row <= 4; row++)
    {
        for (column = 0; column <=4; column++)
            {
                printf ("%d ", my_array[row][column]);
            }
        printf ("\n");
    }

/* Close stream; skip error-checking for brevity of example */
fclose (my_stream);

return 0;
}

```

Ako sve prođe bez problema, izlaz će biti:

```

Initial values of array:
2  4  6  8  10
12 14 16 18 20
22 24 26 28 30
32 34 36 38 40
42 44 46 48 50
Successfully wrote data to file.
Zeroing array...
0  0  0  0  0

```

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Now reading data back in...

Successfully read data from file.

```
2 4 6 8 10
12 14 16 18 20
22 24 26 28 30
32 34 36 38 40
42 44 46 48 50
```

Ukoliko se pokuša pregledati datoteka `my_numbers.dat` koju kreira program, bilo u nekom tekst editoru, ili korišćenjem neke GNU komande (npr `more my_numbers.dat` ili `cat my_numbers.dat`), izlaz će biti nečitljiv, jer su informacije upisane u binarnom formatu, uglavnom nečitljivom za ljudsko oko.

U C++ programskom jeziku za čitanje i upis binarnih podataka korišćenje operatora `<<i>>` kao i metode `getline` nije najefikasnije, pošto podaci nisu upisivani red po red.

Tokovi imaju dve metode specijalno namenjene upisu i čitanju binarnih podataka: `write` i `read`. Prva od njih je metoda klase `ostream` (koju nasleđuje `ofstream`). `read` je metoda klase `istream` (nasleđena od strane `ifstream`). Objekti klase `fstream` poseduju obe ove metode. Prototipi ovih funkcija su:

```
write (memory_block, size);
read (memory_block, size);
```

gde je `memory_block` tipa `char*` (pokazivač na `char`), a reprezentuje adresu niza karaktera sa koje se čitaju podaci ili na koju treba smestiti podatke koji se upisuju. Parametar `size` je celobrojna vrednost koja specificira broj karaktera koje treba pročitati ili upisati iz/u memorijskog bloka.

```
// reading an entire binary file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    streampos size;
    char * memblock;

    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
    if (file.is_open())
    {
        size = file.tellg();
        memblock = new char [size];
        file.seekg (0, ios::beg);
        file.read (memblock, size);
        file.close();

        cout << "the entire file content is in memory";

        delete[] memblock;
    }
    else cout << "Unable to open file";
    return 0;
}
```

U ovom primeru, čitava datoteka je pročitana i smeštena u blok memorije. Prvo, datoteka je otvorena uz indikator

`ios::ate` koji označava da će pokazivač u otvorenoj datoteci biti lociran na kraju datoteke. Na ovaj način, pozivom metode `tellg()`, dobićemo informaciju kolika je veličina datoteke.

Kada smo saznali veličinu datoteke, alociramo memoriju dovoljno veliku da bismo smestili podatke koji se čitaju:

```
memblock = new char[size];
```

Odmah nakon toga, nastavljamo sa postavljanjem pokazivača na sam početak datoteke (otvorili smo datoteku sa pokazivačem na kraju datoteke) a onda čitamo čitavu datoteku i na kraju je zatvaramo:

```
file.seekg (0, ios::beg);  
file.read (memblock, size);  
file.close ();
```

Nakon toga, možemo raditi sa podacima pročitanim iz datoteke, ali naš program samo ispisuje poruku i završava se.

1.5.Pozicija u datoteci

Slično kao što je neophodno zapamtiti stranicu na kojoj se stalo prilikom čitanja knjige, u slučaju rada sa datotekama je potrebno znati poziciju do koje se stiglo prilikom čitanja ili upisivanja u datoteku. Takođe, često može biti korisno imati mogućnost promene trenutne pozicije u okviru datoteke.

Na visokom nivou rada sa datotekama, GNU tretira sve tokove kao nizove karaktera, čak i binarno zapisane tokove, kao u slučaju datoteke `my_numbers.dat` iz prethodnog primera. Ovo znači da je pozicija datoteke zapravo broj koji označava mesto unutar datoteke: pozicija 0 znači da se čita ili piše na mesto prvog karaktera u datoteci, pozicija 522 označava da se to radi na mestu 523-eg karaktera u datoteci i slično.

Osim toga, prilikom čitanja ili upisa u datoteku, pozicija unutar datoteke se uvećava, a osim toga, korišćenjem funkcija za rad sa datotekama na visokom nivou, moguće je promeniti poziciju u datoteci po volji. Svaka datoteka koja omogućava promenu pozicije unutar datoteke na proizvoljan način, naziva se *datoteka sa proizvoljnim pristupom* (random-access file).

Osnovna funkcija visokog nivoa koja se koristi u svrhu saznanja trenutne pozicije je *ftell*. Ova funkcija prihvata jedan jedini parametar, tok koji predstavlja vezu sa prethodno otvorenom datotekom. Funkcija vraća celobrojnu vrednost koja predstavlja poziciju unutar datoteke.

Funkcija koja se koristi kako bi se menjala pozicija je *fseek*. Ova funkcija prihvata tri parametra:

- tok koji predstavlja vezu sa datotekom

- long integer ofset vrednost

- konstanta koja specificira da li je ofset relativan u odnosu na početak datoteke (`SEEK_SET`), trenutnu poziciju (`SEEK_CUR`), ili kraj datoteke (`SEEK_END`). *Fseek* funkcija vraća vrednost 0 ukoliko je uspešno obavljena ova operacija, ili nenultu vrednost u suprotnom.

Takođe, jednostavan makro *rewind* je dostupan i može se koristiti kako bi se pokazivač na poziciju unutar datoteke vratio na početnu vrednost (0). Ovo se radi tako što se tok koji se koristi prilikom rada sa datotekom, jednostavno prosledi makrou *rewind*. U ovom slučaju nema povratne vrednosti. Ova akcija je ekvivalentna pozivanju *fseek* funkcije sa ofsetom 0 i trećim parametrom `SEEK_SET`, sa razlikom u tome što se korišćenjem makroa resetuje eventualna indikacija greške prilikom rada sa tokom i, kao što je rečeno, nema povratne vrednosti.

Primer rada sa ovim funkcijama će biti prikazan kada budemo pokazali kako se radi sa ulazno/izlaznim podacima u maniru karakter po karakter.

Kod C++ programskog jezika, pozicije u datoteci se proveravaju na sličan način kao u slučaju C programskog jezika, ali postoje određene razlike u skladu sa klasama koje se koriste:

`ifstream`, kao i `istream`, čuva interno poziciju preuzimanja, koja predstavlja lokaciju elementa koji će sledeći biti pročitani iz datoteke.

`ofstream`, kao i `ostream`, čuva interno poziciju smeštanja, koja predstavlja lokaciju na koju će sledeći element biti upisan.

Konačno, `fstream`, čuva obe pozicije, preuzimanja i smeštanja, baš kao i `iostream`.

Ove interne pozicije u toku se mogu pročitati i modifikovati korišćenjem sledećih funkcija članica odgovarajućih klasa:

```
tellg() i tellp()
```

Ove dve funkcije članice, koje nemaju parameter, vraćaju vrednost tipa `streampos`, koji je tip zadužen da predstavlja trenutnu poziciju preuzimanja (u slučaju `tellg`) ili poziciju smeštanja (u slučaju `tellp`).

```
seekg() i seekp()
```

Ove metode omogućavaju promenu pozicije preuzimanja i smeštanja. Obe funkcije postoje u dve forme, odnosno mogu imati različit broj argumenata. Prva od formi je:

```
seekg (position);
```

```
seekp (position);
```

Ako se koriste ove funkcije, pokazivač u toku je promenjen na apsolutnu poziciju `position` (izračunatu u odnosu na početak datoteke). Tip prosleđenog parametra je `streampos`, koji je isti kao i tip povratne vrednosti funkcija `tellg` i `tellp`.

Druga forma ovih funkcija podrazumeva dva argumenta:

```
seekg (offset, direction);
```

```
seekp (offset, direction);
```

Korišćenjem ovih funkcija pozicije preuzimanja i smeštanja se postavljaju na relativnu poziciju u odnosu na određene predefinisane pozicije specificirane parametrom `direction`. `offset` je argument tipa `streamoff`. Bilo koja vrednost argumenta `direction` je tipa `seekdir`, koji je enumeracija i određuje poziciju u odnosu na koju se `offset` računa, a može imati bilo koju od narednih vrednosti:

`ios::beg` Ofset se računa u odnosu na početak toka

`ios::cur` Ofset se računa u odnosu na trenutnu poziciju u okviru toka

`ios::end` Ofset se računa u odnosu na poslednju poziciju u okviru toka (kraj toka)

Primer:

```
// obtaining file size
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    streampos begin,end;
    ifstream myfile ("example.bin", ios::binary);
    begin = myfile.tellg();
    myfile.seekg (0, ios::end);
    end = myfile.tellg();
    myfile.close();
    cout << "size is: " << (end-begin) << " bytes.\n";
    return 0;
}
```

Obratite pažnju na tip varijabli `begin` i `end` korišćenih u programu. Kao što smo već rekli, `streampos` je specifičan tip podataka korišćen za rad sa baferima i pozicioniranju unutar datoteka, a to je tip povratne vrednosti funkcije `file.tellg()`. Vrednosti ovog tipa se mogu oduzimati jedna od druge bezbedno, a takođe se mogu i konvertovati u celobrojni tip dovoljno velik da smesti vrednost promenljive (odnosno veličine datoteke).

1.6. Baferovanje tokova

Prilikom upisa karaktera u tok, oni se ne smeštaju u datoteku karakter po karakter, čim se pojave u toku. Umesto toga, oni se akumuliraju u baferu, a zatim iz bafera upisuju u datoteku u blokovima kada se određeni uslovi ispune (Bafer u ovom smislu je prostor memorije računara koji se koristi za privremeno smeštanje podataka). Slično, prilikom čitanja karaktera iz toka, oni se najpre baferuju, tj. smeštaju najpre u bafer.

Važno je napomenuti kako baferovanje zaista radi, u suprotnom može doći do neočekivanog ponašanja programa, čitanja ili upisa podataka kada to nije očekivano ponašanje i slično. Baferovanje se može izbeći u potpunosti, ali samo korišćenjem rutina niskog nivoa, ne visokog nivoa.

Postoje tri osnovna tipa baferovanja koji su od interesa:

- Bez baferovanja - Kada se karakteri upisuju u ne-baferovani tok, operativni sistem ih upisuje u datoteku čim je to moguće
- Linijsko baferovanje - Prilikom upisa karaktera u linijski baferovan tok, operativni sistem će ih upisati u datoteku tek kada naiđe na novi red (*newline* karakter)
- Puno baferovanje - Kada se upisuju karakteri ovom metodom, operativni sistem upisuje podatke u datoteku u blokovima proizvoljne veličine.

Većina tokova koristi puno baferovanje i ovo je najčešće najbolje rešenje. Ipak, tokovi koji se koriste za komunikaciju sa eksternim uređajima su najčešće linijski baferovani, što se samim tim i postavlja kao podrazumevano ponašanje nakon otvaranja (ovo se odnosi i na *stdin* i *stdout*).

Činjenica da su *stdin* i *stdout* linijski baferovani je zgodna jer su uglavnom podaci koji se šalju putem njih terminirani karakterom koji predstavlja novi red (zavisi od operativnog sistema). U cilju obezbeđivanja da se podaci koji se čitaju ili upisuju odmah, koristi se *fflush* funkcija. Na ovaj način se forsira premeštanje karaktera iz bafera u datoteku, ukoliko to već nije urađeno ranije. Ova funkcija se koristi tako što joj se prosledi tok kojem želimo da ispraznimo bafer. Povratna

vrednost je 0 ako je sve prošlo kako treba, EOF (makro definisan u okviru GNU C biblioteke) ako je došlo do greške prilikom upisa. Ipak, korišćenje ove funkcije nije obavezno: izlaz će se *flush*-ovati kada se upisuje u već pun bafer, kada se zatvara tok (zatvara se datoteka koju dati tok predstavlja), kada se program završava, kada je newline karakter upisan u linijski baferovan tok.

1.7. End-of-file (EOF) i funkcije za informacije o greškama

Ukoliko se datoteka pročitana do kraja (tj. pozicija u datoteci stigne do kraja datoteke), specijalan indikator EOF će se postaviti na vrednost TRUE. Vrednost ovog indikatora može biti proverena korišćenjem *feof* funkcije. Ova funkcija prihvata samo jedan parametar (tok) i vraća vrednost TRUE (nenultu vrednost) ako se došlo do kraja datoteke, ili FALSE u suprotnom.

Još jedan indikator, indikator greške, se koristi da signalizira da se greška pojavila prilikom prethodne operacije na toku. Ovaj indikator ima vrednost TRUE ako je greška prisutna, odnosno FALSE u suprotnom. Vrednost ovog indikatora se može proveriti za dati tok korišćenjem *ferror* funkcije. Ova funkcija prima tok kao parametar i vraća vrednost indikatora greške (TRUE ili FALSE).

Nažalost, *ferror* funkcija nam neće javiti koja greška je nastala ili gde je nastala, jedino da je došlo do greške. Da bi se dobilo više informacija o samoj grešci, potrebno je proveriti globalnu sistemsku promenljivu *errno*.

Moguće je resetovati indikator greške, kao i EOF indikator. Da bi se to uradilo, jednostavno se prosledi tok funkciji *clearerr*; ovo će oba indikatora postaviti na vrednost 0. Ova funkcija ne vraća nikakvu vrednost.

Nije preporučljivo samo resetovati indikator greške i pokušati operaciju na toku nakon toga. Usled baferovanja, na ovaj način je moguće izgubiti podatke prilikom upisa, ili jednostavno dobiti pogrešne podatke iz datoteke prilikom čitanja. Pre nego što se ponovo pokuša prethodno neuspešna operacija na datoteci, potrebno je najpre podesiti poziciju u datoteci na pravu vrednost. Međutim, većina (ako ne i sve) grešaka će rezultovati istom greškom ukoliko se ponovi ista operacija nakon neuspelog pokušaja. Samim tim, uglavnom je rešenje koje se koristi da se vrati poruka o grešci korisniku prilikom izlaska iz programa.

Primer rada sa ovim funkcijama će biti prikazan kada budemo pokazali kako se radi sa ulazno/izlaznim podacima u maniru karakter po karakter.

U C++ programskom jeziku, provera stanja toka vrši se korišćenjem sledećih metoda pozvanih na toku koji se koristi:

- `bad()` vraća true ako je operacija čitanja ili upisa bila neuspešna. Na primer, u slučaju da pokušamo da upisujemo u datoteku koja nije otvorena za upisivanje.
- `fail()` vraća true u istim slučajevima kao i `bad()`, ali takođe i u slučaju kada se desi greška kao posledica formatiranja, kao na primer tekstualni karakter prihvaćen iz toka u slučaju kada smo očekivali da pročitani podatak bude celi broj (integer).
- `eof()` vraća true ako se čitanjem datoteke, koja je otvorena za čitanje, došlo do njenog kraja.
- `good()` je indikator koji generalizuje sve prethodne: on vraća false u svim slučajevima u kojima gornji indikatori vraćaju true. Treba primetiti da `bad()` i `good()` nisu potpuno suprotni indikatori jer `good` uzima u obzir više stanja i indikatora odjednom.

2. Unos i čitanje stringova

U nastavku će biti pokazane rutine koje na visokom nivou omogućavaju čitanje stringova kao i unos stringova. Najpre će biti reči o dve metode (`puts` i `fputs`) koje su veoma bezbedne za korišćenje, ali ćemo navesti i metode koje treba izbegavati (`gets`) do i donekle bezbednijeg `fgets`, kao i `getline` i `getdelim` (dve specifične i preporučljive metode za GNU sisteme koje su apsolutno ekstremno za korišćenje).

2.1.Neformatirani ispis stringa

Ova funkcije su veoma bezbedne za korišćenje.

puts

Najzgodnija funkcija za ispis jednostavnih poruka na standardnom izlazu. Još je jednostavnija od printf, jer nije potrebno dodavati karakter za novi red, puts to radi automatski.

```
puts ("Hello, multiverse.");
```

Iako je bezbedna i jednostavna, nije baš previše fleksibilna.

fputs

Funkcija fputs ("file put string") je slična puts funkciji, osim što prihvata i drugi parametar, koji predstavlja tok u koji se upisuje string prosleđen kroz prvi parametar. Ova funkcija ne dodaje newline karakter. Vraća EOF ukoliko je došlo do greške, u suprotnom vraća nenegativnu celobrojnu vrednost.

U primeru ispod, kreira se tekstualna datoteka i koristi se fputs kako bi se upisao tekstu u nju. Takođe je demonstrirana i fflush funkcija.

```
#include <stdio.h>

int main()
{
    FILE *my_stream;
    char my_filename[] = "snazzyjazz.txt";
    int flush_status;

    my_stream = fopen (my_filename, "w");
    fputs ("If it's not too late... make it a cheeseburger.\n", my_stream);

    /*
     * Since the stream is fully-buffered by default, not line-buffered,
     * it needs to be flushed periodically. We'll flush it here for
     * demonstration purposes, even though we're about to close it.
     */
    flush_status = fflush (my_stream);
    if (flush_status != 0)
    {
        puts ("Error flushing stream!");
    }
    else
    {
        puts ("Stream flushed.");
    }

    /* Close stream; skip error-checking for brevity of example */
    fclose (my_stream);

    return 0;
}
```

2.2.Formatirani ispis stringova

Funkcije u ovoj sekciji omogućavaju formatirani ispis stringova. One su, generalno, bezbedne za korišćenje.

Formatirani izlaz podrazumeva tekstualni izlaz preko funkcija kao što su printf ili fprintf. Ove funkcije primaju kao argument string koji sadrži specijalne sekvence karaktera kao što je %d (koji daje indikaciju da sledi celobrojna vrednost). Nakon ovog stringa, ostali parametri odgovaraju specijalnim karakterima, a funkcija kombinuje ove argumente kako bi rezultovala formatiranim tekstualnim izlazom.

U narednoj sekciji (nekoliko njih) diskutovaće se četiri funkcije za formatirani ispis. Osnovna, `printf`, prikazuje ispis na standardnom izlazu. `Fprintf` je funkcija visokog nivoa koja šalje ispis ka predefinisanim toku, `sprintf` "ispisuje" izlaz u novi string, dok je `asprintf` bezbednija funkcija sa istom ulogom.

fprintf

Funkcija `fprintf` ("file print formatted") je identična funkciji `printf`, osim što je prvi parametar tok u koji se šalje izlaz.

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

int main()
{
    int my_integer = -42;
    unsigned int my_ui = 23;
    float my_float = 3.56;
    double my_double = 424242.171717;
    char my_char = 'w';
    char my_string[] = "Pardon me, may I borrow your nose?";

    FILE *my_stream;
    char my_filename[] = "snazzyjazz.txt";
    my_stream = fopen (my_filename, "w");

    fprintf (my_stream, "Integer: %d\n", my_integer);
    fprintf (my_stream, "Unsigned integer: %u\n", my_ui);

    fprintf (my_stream, "The same, as hexadecimal: %#x %#x\n", my_integer, my_ui);

    fprintf (my_stream, "Floating-point: %f\n", my_float);
    fprintf (my_stream, "Double, exponential notation: %17.11e\n", my_double);

    fprintf (my_stream, "Single character: %c\n", my_char);
    fprintf (my_stream, "String: %s\n", my_string);

    errno = EACCES;
    fprintf (my_stream, "errno string (EACCES): %m\n");

    /* Close stream; skip error-checking for brevity of example */
    fclose (my_stream);

    return 0;
}
```

asprintf

`asprintf` (mnemonic: "allocating string print formatted") je identična kao i `printf`, osim što je prvi parametar string u koji se šalje izlaz. Ova funkcija terminira string null karakterom. Vraća broj karaktera koji su smešteni u stringu, ne uključujući terminirajući null karakter.

Funkcija `asprintf` je gotovo identična jednostavnijoj `sprintf` funkciji, ali je mnogo bezbednija za korisničenje, jer dinamički alokira string u koji šalje izlaz, tako da prilikom upisa u string nikada neće doći do prekoračenja opsega. Prvi parametar je pokazivač na string promenljivu koji je, samim tim, tipa `char **`. Povratna vrednost je broj karaktera koji su alocirani za bafer, ili negativna vrednost ukoliko se desila greška. Sledeći primer pokazuje korišćenje ove funkcije (obratiti pažnju da pre poziva funkcije nije alocirani prosto za niz `my_string`, jer će to uraditi sama funkcija `asprintf`):

```
#include <stdio.h>

int main()
{
    char *my_string;
```

```

asprintf (&my_string, "Being %d is cool, but being free is best of all.", 4);
puts (my_string);

return 0;
}

```

2.3.Unos stringova

Funkcije navedene u ovom poglavlju su veoma bezbedne za korišćenje.

getline

Funkcija *getline* je preferirana funkcija za čitanje ulaza, kako iz datoteka, tako i od strane standardnog ulaza. Ostale standardne funkcije koje se koriste u ovu svrhu, uključujući *gets*, *fgets* i *scanf* su nepouzdana.

Ova funkcija čita ceo red iz toka, sve do (i uključujući) newline karakter. Ova funkcija prima tri parametra:

- pokazivač na blok alociran pomoću *malloc* ili *calloc*. Ovaj parametar je tipa *char *** i sadržaće pročitane liniju kada se funkcija izvrši do kraja.

- Drugi parametar je pokazivač na promenljivu tipa *size_t* koji specificira kolika je veličina u bajtovima bloka memorije na koju pokazuje prvi parametar.

- Treći parametar je tok koji se koristi i iz koga se čitaju podaci.

Pokazivač na blok memorije alocirane za *getline* funkciju je samo predlog. *Getline* funkcija automatski uvećava blok alocirane memorije po potrebi, korišćenjem *realloc* funkcije, tako da nikako neće nedostajati prostora, usled čega je ova funkcija bezbedna za korišćenje. Ne samo to, *getline* će takođe saopštiti novu veličinu alociranog bloka ažuriranjem drugog parametra (zbog toga je i prosleđen preko pokazivača). Ukoliko dođe do greške, kao što je kraj datoteke koji je dostignut pre nego što su ikakvi podaci pročitani, *getline* vraća vrednost -1. U drugom slučaju, prvi parametar sadrži pokazivač na string koji se sastoji od pročitane linije a *getline* funkcija vraća broj pročitanih karaktera (sve do i uključujući newline karakter ali ne i null terminator). Tip povratne vrednosti je *size_t*.

Iako je prvi argument pointer na string (*char ***), nije moguće tretirati ga kao "običan" string, pošto može sadržati null karaktere pre finalnog null terminatora koji označava kraj stringa. Povratna vrednost funkcije u ovom slučaju omogućava razlikovanje null karaktera u okviru primljenom stringa (kao deo primljene poruke) tako što povratna vrednost ukazuje na broj primljenih karaktera. Svi karekteri u okviru bloka koji se nalaze na pozicijama manjim od povratne vrednosti funkcije, zapravo su deo primljene linije, svi karakteri veći od nje nisu.

Evo primera koda koji demonstrira kako se koristi *getline* u cilju čitanja linije teksta sa tastature na bezbedan način. Bezbednost funkcije se može proveriti tako što se pošalje više od 100 karaktera. Treba obratiti pažnju da *puts*, koji se koristi za prikaz primljenih karaktera, neće biti adekvatan ukoliko među primljenim karakterima ima null karaktera (tretiraće ih kao kraj stringa). Ipak, obzirom da nije baš jednostavno poslati null karakter preko tastature, u ovom slučaju to je bezbedno, ali treba biti oprezan ako se na ovaj način preuzimaju podaci iz datoteke.

```

#include <stdio.h>

int main()
{
    int bytes_read;
    int nbytes = 100;
    char *my_string;

    puts ("Please enter a line of text.");

    /* These 2 lines are the heart of the program. */
    my_string = (char *) malloc (nbytes + 1);
    bytes_read = getline (&my_string, &nbytes, stdin);

```

```

if (bytes_read == -1)
{
    puts ("ERROR!");
}
else
{
    puts ("You typed:");
    puts (my_string);
}

return 0;
}

```

Getdelim

Funkcija `getdelim` predstavlja generalizaciju `getline` funkcije; dok `getline` završava sa čitanjem ulaza prilikom prvog newline karaktera, `getdelim` funkcija omogućava korisnika da specificira neki drugi delimiter koji će se koristiti umesto newline karaktera. Zapravo, `getline` funkcija jednostavno poziva `getdelim` i specificira da je delimiter newline karakter.

Sintaksa za `getdelim` funkciju je gotovo identična kao u slučaju `getline` funkcije, sa razlikom što treći parametar specificira delimiter, a četvrti predstavlja tok koji se koristi za čitanje.

Na primeru od gore, u kome je pokazana upotreba `getline` funkcije, program bi se isto ponašao ukoliko bismo zamenili liniju

```

bytes_read = getline (&my_string, &nbytes, stdin);
sa linijom

bytes_read = getdelim (&my_string, &nbytes, '\n', stdin);

```

2.4.Formatirani unos stringova

Formatiran i unos stringova je suprotno od formatiranih funkcija za prikaz stringa. Nasuprot `printf` i sličnim funkcijama o kojima je bilo reči ranije, `scanf` i slične funkcije parsiraju formatirani ulaz. Slično kao u slučaju suprotnih funkcija, svaka prihvata kao parametar šablon stringa koji sadrži specifične karaktere za konverziju. U slučaju `scanf` i pridruženih funkcija, ipak, konverzija je namenjena kako bi poklopio šablon u ulaznom stringu (celobrojne vrednosti, floating-point brojevi, sekvence karaktera i slično) i čuva prepoznate vrednosti u prosleđenim promenljivama.

sscanf

Ova funkcija prihvata string iz koga čita ulaz, zatim, u maniru sličnom `printf` funkciji, preuzima šablomn string i niz prosleđenih argumenata, pokušava da prepozna string koji se koristi kao ulaz u datom šablon stringu, pri čemu koristi konverziju isto kao u slučaju `printf` funkcije.

Funkcija `sscanf`, je slična kao zastarela `scanf` funkcija, osim što u slučaju `sscanf` funkcije, na mestu prvog parametra specificira se string iz koga se čita, dok se u slučaju `scanf` funkcije uvek čita standardni ulaz. Dostizanje kraja stringa se tretira kao i EOF uslov.

Evo primera:

```

sscanf (input_string, "%as %as %as", &str_arg1, &str_arg2, &str_arg3);
Ukoliko se string koji sscanf funkcija skenira koristi i kao jedan od argumenata, rezultat je potpuno nepredvidiv i ovo
ne bi trebalo nikako raditi:

```

```

sscanf (input_string, "%as", &input_string);
Evo primera dobrog koda koji parsira ulaz pomoću sscanf funkcije. Traži od korisnika da unese tri celobrojne vrednosti

```


razdvojene whitespace karakterom, nakon čega čita proizvoljno dugačku liniju teksta korišćenjem geline funkcije. Posle toga, proverava se da li su tačno tri argumenta obezbeđena sscanf funkciji. Ukoliko pročitana linija sadrži podatke koji nisu traženi (npr. sadrži floating-point brojeve umesto celobrojnih), program ispisuje poruku o grešci i traži od korisnika da ponovi unos. Upravo ova fleksibilnost unosa, kao i velika jednostavnost oporavka od greške, čini ovu kombinaciju getline/sscanf superiornu u odnosu na samo scanf. Dakle, uvek treba koristiti ovakvu kombinaciju getline/sscanf umesto scanf, ukoliko je moguće.

```
#include <stdio.h>
int main()
{
    int nbytes = 100;
    char *my_string;
    int int1, int2, int3;
    int args_assigned;

    args_assigned = 0;

    while (args_assigned != 3)
    {
        puts ("Please enter three integers separated by whitespace.");
        my_string = (char *) malloc (nbytes + 1);
        getline (&my_string, &nbytes, stdin);
        args_assigned = sscanf (my_string, "%d %d %d", &int1, &int2, &int3);
        if (args_assigned != 3)
            puts ("Input invalid!");
    }

    printf ("Thanks!%d %d %d", int1, int2, int3);

    return 0;
}
```

Šablon string koji koristi sscanf i slične funkcije je donekle slobodna forma u odnosu na one koje koristi printf. Na primer, većina specifikatora konverzija ignoriše prazna mesta na početku. Osim toga, nije moguće specificirati preciznost kod sscanf konverzija kao što se to radi kod printf funkcije.

Još jedna značajna razlika između sscanf i printf leži u činjenici da argumenti prosleđeni sscanf funkciji moraju biti pokazivači; ovo omogućava sscanf funkciji da vrati vrednosti kroz promenljive na koju ovi pokazivači referenciraju. Ukoliko se zaboravi proslediti pokazivač sscanf funkciji, moguće je dobiti neke čudne i neočekivae greške, a svakako može lako doći do toga; stoga, ovo je jedna od prvih stvari koje treba proveriti ukoliko sscanf funkcija počne da se ponaša “čudno”.

Šablon korišćen od strane sscanf može da sadrži proizvoljan broj praznih mesta (space-ova), proizvoljan broj specifikatora konverzija koji počinju sa %. Prazno polje u šablon stringu odgovara nijednom ili više praznih polja u ulaznom stringu. Sa druge strane, svi karakteri koji nisu prazna polja, moraju tačno da se poklope sa ulaznim stringom, u suprotnom se smatra da nema poklapanja šablon stringa i ulaznog stringa. Na primer, šablon string “foo “ poklapa se sa “foo” ili “foo “ ali ne i sa “food”.

Ako se kreira specifikator konverzije koji nije validan, ili ukoliko se ne navede dovoljno argumenata za sve navedene specifikatore konverzija, kod može da rezultuje neočekivanim rezultatima, tako da treba biti veoma oprezan. Suvišni argumenti neće dovesti do greške, već će jednostavno biti ignorisani.

Specifikatori konverzija počinju sa znakom % i terminiraju se jednim od karaktera iz naredne liste:

c

Poklapa se sa fiksnim brojem karaktera. Ako se specificira polje za maksimalnu širinu, ovoliko će karaktera biti poklopljeno, u suprotnom %c se podudara samo sa jednim karakterom. Ova konverzija se ne odnosi na null karakter koji terminira tekst koji se čita, kao što radi %s konverzija. Takođe, ne preskače prazna polja (space karaktere), već čita tačno broj karaktera koji je zahtevan, ili generiše grešku prilikom konverzije ukoliko to nije moguće

d

Podudara se sa opciono označenim celobrojnim brojem, a može da uključuje i sledeće:

1. Opcioni plus minus znak (+ ili -).
2. Jedna ili više decimalnih cifara.

Treba obratiti pažnju da %d i %i nisu sinonimi za scanf, kao što su za printf.

e

Poklapa se sa opciono označenim floating-point brojem, uključujući i:

1. Opcioni plus minus znak (+ ili -).
2. Floating-point broj u decimalnom ili heksadecimalnom formatu.
 - Decimalni format je sekvenca jednog ili više decimalnih cifara, koje opciono sadrže i karakter decimalne tačke, praćeno opciono delom koji se odnosi na eksponent a sadrži slovo e ili E, opciono plus ili minus znak i opet sekvencu decimalnih cifara
 - Heksadecimalni format je 0x ili 0X, praćen sekvencom jedne ili više heksadecimalnih cifara, opciono sadržavajući karakter za decimalnu tačku, praćen opciono binarno-eksponencijalnim delom koji sadrži karaktere p ili P, opciono plus ili minus znak i nakon toga sekvencu cifara.

E

Isto kao e.

f

Isto kao e.

g

Isto kao e.

G

Isto kao e.

i

Poklapa se sa opciono označenim celobrojnim brojem, koji sadrži:

1. Opciono plus ili minus znak (+ ili -).
2. String karaktera koji predstavljaju neoznačeni celobrojni broj:
 - Ako string počinje sa 0x ili 0X, broj se podrazumeva da je u heksadecimalnom formatu, dok ostatak stringa mora sadržati heksadecimalne cifre
 - U suprotnom, ako string počinje sa 0, broj se smatra da je u oktalanom formatu (osnova brojnog sistema 8) a ostatak stringa mora sadržati oktalne cifre
 - U suprotnom, broj se podrazumeva da je u decimalnom formatu, dok ostatak stringa mora sadržati decimalne cifre

Važno je primetiti da %d i %i nisu sinonimi kao u slučaju printf funkcije. Sa printf funkcijom, celobrojne vrednosti se mogu štampati u ovom formatu korišćenjem # indikatorskog karaktera, ili %x ili %d izlaznih konverzija.

s

Poklapa se sa stingom karaktera od kojih ni jedan nije space. Preskače se inicijalni whitespace karakter, ali se zaustavlja kada naiđe na sledeći prilikom čitanja. Null karakter se smešta na kraj pročitnog teksta, kako bi se označio kraj stringa

x

Podudara se sa neoznačenim celobrojnim vrednostima u heksadecimalnom formatu. String sa kojim se poredi mora početi sa 0x ili 0X, dok ostatak stringa moraju sačinjavati heksadecimalne cifre.

X

Isto kao x.

[

Podudara se sa stringom koji sadrži proizvoljne karaktere. Na primer, %12[0123456789] znači da se pročita stinga sa maksimalno 12 karaktera koji sadrži karaktere iz skupa 123456789, drugim rečima dvanaestocifarni decimalni broj. Ukoliko se pojavi znak “-” on predstavlja opseg, pa se primer od gore može zapisati i kao %12[0-9]. Ukoliko je prvi karakter unutar zagrade “^” to znači da se čita string koji ne sadrži karaktere izlistane u zagradi. Dakle, %12[^0-9] znači čitati 12 karaktera od kojih ni jedan nije decimalna cifra.

%

Poklapa se sa znakom procenta.

Između znaka procenta i ulaznog karaktera za konverziju, moguće je postaviti kombinacije sledećih modifikatora, u sekvenci (ovo se ne odnosi na %% konverziju)

- Opcioni * indikator. Ovaj indikator specificira da je neophodno poklapanje između specifikatora konverzije i ulaznog toka, ali da vrednost ne treba da bude dodeljena argumentu
- Opcioni a indikator, koji je validan samo za string konverzije. Ovo je GNU ekstenzija scanf funkcije koja zahteva da se alocira bafer dovoljne veličine kako bi bezbedno smestio string koji je pročitao.
- Opcioni ' indikator. Ovaj indikator specificira da će pročitani broj biti grupisan u skladu sa pravilima trenutno podešenim na korisničkom sistemu. Na primer, u SAD-u ovo znači da će 1,000 biti pročitano kao jedna hiljada
- Opciona decimalna celobrojna vrednost koja specificira maksimalnu širinu polja. Funkcija scanf će se zaustaviti sa čitanjem karaktera iz ulaznog toka ili nakon maksimalnog broja pročitanih karaktera ili kada se naiđe na prvi ne-podudarajući karakter, zavisi šta se prvo desi. Odbačeni početni whitespace karakteri se ne računaju u ovu maksimalnu širinu polja, kao ni null terminator smešten da označi kraj stringa
- Opcioni modifikator tipa iz tabele ispod (podrazumevani tip odgovarajućih argumenata je int * za %d i %i konverzije, unsigned int * za %x i %X konverzije i float * za %d i njegove sinonime)

h

Specificira da argument kojem se čita vrednost treba da bude tipa short int* ili unsigned short int*. Ovaj modifikator je validan za %d i %i konverzije.

l

Za %d i %i konverzije, specificira da je argument kojem se pročitana vrednost pridružuje long int* ili unsigned long int*. Za %e konverzije i njene sinonime, specificira da je argument tipa double *.

L

Za %d i %i konverzije, specificira da je argument kojem se pročitana vrednost pridružuje long long int* ili unsigned long long int*. Na sistemima koji nemaju ovakav tip promenljivih, efekat je isti kao da je korišćen modifikator l. Za %e konverzije i njene sinonime, specificira da je argument tipa long double *.

ll

Isto kao L za %d i %i konverzije.

q

Isto kao L za %d i %i konverzije.

z

Specificira da je argument kojem se dodeljuje pročitana vrednost tipa size_t. Size_t je tip koji je korišćen kako bi specificirao veličine blokova memorije i mnoge funkcije koje su navedene ovde ga koriste. Ovaj modifikator je validan za %d i %i konverzije.

U C++ programskoj jeziku, rad sa tekstualnim datotekama podrazumeva rad sa tokovima otvorenim bez indikatora ios::binary. Upis u tekstualne datoteke vrši se na sličan način kao sa tokom cout:

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    ofstream myfile ("example.txt");
    if (myfile.is_open())
    {
        myfile << "This is a line.\n";
        myfile << "This is another line.\n";
        myfile.close();
    }
}
```

```

    }
    else cout << "Unable to open file";
    return 0;
}

```

Čitanje iz datoteke se vrši slično kao i korišćenjem `cin`:

```

// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
    string line;
    ifstream myfile ("example.txt");
    if (myfile.is_open())
    {
        while ( getline (myfile,line) )
        {
            cout << line << '\n';
        }
        myfile.close();
    }

    else cout << "Unable to open file";

    return 0;
}

```

U ovom primeru, čita se tekstualna datoteka i ispisuje se sadržaj na ekranu. While petlja čita red po red iz tekstualne datoteke korišćenjem metode `getline`. Povratna vrednost ove metode je referenca na sam taj tok, koji onda može da se evaluira kao Boolean izraz (kao što se to radi ovde u while petlji) i dobiće se vrednost `true` ako je tok spreman za naredno čitanje, ili `false` ako smo došli do kraja datoteke ili je došlo do neke greške prilikom čitanja.

Kada je potrebno prilikom čitanja iz datoteke odmah konvertovati podatak iz string-a u neki drugi tip to se vrši na sledeći način:

```

#include <iostream>
#include <fstream>
using namespace std;

int main () {
    double number;
    ifstream myfile ("example.txt"); //neka je u example.txt upisan tekst 3.14
    if (myfile.is_open())
    {
        myfile >> number; //podatak iz datoteke se čita u promenljivu tipa double
        cout << "Procitan broj je: " << number;
        myfile.close();
    }

    else cout << "Unable to open file";

    return 0;
}

```

Složenije konverzije se mogu vršiti korišćenjem `std::istringstream` objekta koji od string-a „pravi“ tok:

```

#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main()
{
    std::string str = "10.3 12 test";

    istringstream ss(str);
    float d;
    int i;
    string s;

    ss >> d; //podrazumevani delimiter je space, pa će pročitati sve do prvog " "
    std::cout << "Procitan broj je: " << d << std::endl;
    ss >> i;
    std::cout << "Procitan int je: " << i << std::endl;
    ss >> s;
    std::cout << "Procitan string je: " << s << std::endl;
}

```

3. Unos i ispis jednog karaktera

U ovom poglavlju su pokrivene funkcije koje se koriste za unos i čitanje pojedinačnih karaktera kako sa standardnog ulaza/izlaza tako i od strane datoteka.

getchar

Ukoliko je potrebno čitati jedan karakters sa standardnog ulaza može se koristiti getchar funkcija. Ova funkcija nema parametre, čita naredni karakter od strane standardnog ulaza stdin kao unsigned char i vraća ovu vrednost, konvertovanu u celobrojnu vrednost. Evo jednostavnog primera:

```

#include <stdio.h>

int main()
{
    int input_character;

    printf("Hit any key, then hit RETURN.\n");
    input_character = getchar();
    printf ("The key you hit was '%c'.\n", input_character);
    printf ("Bye!\n");

    return 0;
}

```

Treba zapaziti da, obzirom da je stdin linijski baferovan, getchar neće vratiti vrednost sve dok se ne pritisne <RETURN> dugme. Međutim, getchar i dalje čita samo jedan karakter od strane stdin, tako da ukoliko se unese hellohellohello na ulazu, program iznad će i dalje prihvatiti samo jedan karakter:

```

The key you hit was 'h'.
Bye!

```

putchar

Ukoliko je potrebno odštampati jedan karakter na standardni izlaz, moguće je koristiti funkciju putchar. Ona prima jedan parametar koji sadrži karakter koji treba prikazati (argument može biti naveden pod jednostrukim navodnicima, kao u primeru iznad) i šalje dati karakter na stdout. Ukoliko se desi greška prilikom upisa, putchar vraća EOF; u suprotnom vraća celobrojnu vrednost koja joj je bila prosleđena.

Evo kratkog primera kako se koristi putchar. U primeru se štampa X, nakon toga razmak i na kraju linija od 10 uzvičnika.

Nakon toga sledi newline karakter tako da se sledeći shell prompt neće pojaviti u istoj liniji. Treba primetiti kako se koristi for petlja-na ovaj način se putchar može koristiti za prikaz jednog karaktera, ali više puta.

```
#include <stdio.h>

int main()
{
    int i;

    putchar ('X');
    putchar (' ');
    for (i=1; i<=10; i++)
    {
        putchar ('!');
    }
    putchar ('\n');

    return 0;
}
```

getc and fgetc

Ukoliko postoji potreba čitanja jednog karaktera iz toka drugačijeg od standardnog, moguće je koristiti getc funkciju. Ova funkcija je veoma slična getchar funkciji, ali prihvata argument koji specificira tok iz koga se čita. Nakon toga se čita sledeći karakter iz specificiranog toka kao unsigned char, i vraća njegovu vrednost, konvertovanu u integer. Ukoliko se pojavi greška prilikom čitanja, ili se došlo do kraja datoteke, getc vraća EOF.

Evo primera koda u kome se koristi getc. Ovaj primer kreira datoteku snazzyjazz.txt pomoću fopen, upisuje u njega alfabet velikim slovima zajedno sa newline karakterom korišćenjem fprintf, nakon čega čita poziciju u datoteci pomoću ftell i uzima karakter pomoću getc. Nakon toga se skače na poziciju 25 korišćenjem fseek i ponavlja se proces, pokušava se čitanje nakon dostizanja kraja datoteke i raportira sa EOF status pomoću feof. Takođe se generiše greška nakon pokušaja upisa u read-only tok. Raportira se status greške pomoću ferror, vraća se na početak datoteke pomoću rewind i prikazuje se prvi karakter. Na kraju, pokušava se zatvoriti datoteka i ispisuje se status poruka.

```
#include <stdio.h>

int main()
{
    int input_char;
    FILE *my_stream;
    char my_filename[] = "snazzyjazz.txt";
    long position;
    int eof_status, error_status, close_error;

    my_stream = fopen (my_filename, "w");
    fprintf (my_stream, "ABCDEFGHIJKLMNOPQRSTUVWXYZ");

    /* Close stream; skip error-checking for brevity of example */
    fclose (my_stream);

    printf ("Opening file...\n");
    my_stream = fopen (my_filename, "r");
    position = ftell (my_stream);
    input_char = getc (my_stream);
    printf ("Character at position %d = '%c'.\n\n", position, input_char);

    printf ("Seeking position 25...\n");
    fseek (my_stream, 25, SEEK_SET);
    position = ftell (my_stream);
    input_char = getc (my_stream);
    printf ("Character at position %d = '%c'.\n\n", position, input_char);
}
```

```

printf ("Attempting to read again...\n");
input_char = getc (my_stream);
eof_status = feof (my_stream);
printf ("feof returns %d.\n\n", eof_status);

error_status = ferror (my_stream);
printf ("ferror returns %d.\n", error_status);
printf ("Attempting to write to this read-only stream...\n");
putc ('!', my_stream);
error_status = ferror (my_stream);
printf ("ferror returns %d.\n\n", error_status);

printf ("Rewinding...\n");
rewind (my_stream);
position = ftell (my_stream);
input_char = getc (my_stream);
printf ("Character at position %d = '%c'.\n", position, input_char);

close_error = fclose (my_stream);

/* Handle fclose errors */
if (close_error != 0)
{
    printf ("File could not be closed.\n");
}
else
{
    printf ("File closed.\n");
}

return 0;
}

```

Postoji još jedna funkcija u GNU C biblioteci koja se zove `fgetc`. Ona je identična `getc` funkciji, sa izuzetkom što `getc` je obično implementirana kao makro funkcija koja je izuzetno optimizovana, tako da je preferirana u većini situacija. U situacijama kada se čita sa standardnog ulaza, `getc` je podjednako brza kao i `fgetc`, obzirom da korisnik relativno sporo kuca karaktere u poređenju sa brzinom kojom računari mogu da čitaju ulaz, međutim kada se čita iz toka koji nije u interakciji sa korisnikom, `fgetc` je verovatno bolja opcija.

putc and fputc

Ukoliko je potrebno ispisivati jedan karakter u tok različit od standardnog, moguće je koristiti `putc` funkciju. Ova funkcija je veoma slična `putchar`, ali prihvata argument koji specificira tok u koji se upisuje. Prihvata jedan celobrojni podatak koji sadrži karakter (moguće je da se prosledi sa jednostrukim navodnicima), a zatim štelje karakter specificiranom toku. Ukoliko se pojavi greška prilikom upisa, `putc` vraća EOF. U suprotnom, vraća celobrojnu vrednost koja joj je prosleđena.

Sledeći primer pokazuje kreiranje tekstualne datoteke `snazzyjazz.txt`. Nakon toga se u nju upisuje X, razma a onda linija od deset uzastopnih uzvičnika, praćena karakterom za novi red, korišćenjem `putc` funkcije.

```

#include <stdio.h>

int main()
{
    int i;
    FILE *my_stream;
    char my_filename[] = "snazzyjazz.txt";

    my_stream = fopen (my_filename, "w");

    putc ('X', my_stream);
    putc (' ', my_stream);
    for (i=1; i<=10; i++)
    {

```

```

    putc ('!', my_stream);
}
putc ('\n', my_stream);

/* Close stream; skip error-checking for brevity of example */
fclose (my_stream);

return 0;
}

```

Postoji još jedna funkcija u okviru GNU C biblioteke koja se zove `fputc`. Ona je identična `putc` funkciji, sa razlikom da je `putc` implementirana kao makro funkcija koja je izuzetno optimizovana, pa je treba preferirati kad god je to moguće.

ungetc()

Prilikom svakog čitanja karaktera iz toka korišćenjem funkcija kao što je `getc`, pozicija u datoteci se uvećava za 1. Moguće je uraditi inverznu operaciju funkcijom `ungetc`, koja vraća poziciju u datoteci nazad za jedan bajt, i poništava poslednju operaciju čitanja karaktera iz datoteke.

Namena ovoga je da se indikator pozicije ostavi na korektnom mestu. Programi mogu da pogledaju šta se nalazi ispred, saznaju koji će karakter biti pročitani sledeći, nakon čega će resetovati poziciju sa `ungetc` funkcijom.

Na GNU sistemima nije moguće pozvati `ungetc` dva puta uzastopno bez čitanja barem jednog karaktera između; drugim rečima, GNU podržava samo vraćanje jednog karaktera.

Vraćanje karaktera nazad ne menja datoteku kojoj se pristupa uopšte; `ungetc` jedino ima uticaja na bafer toka, ne na datoteku. Ukoliko je `fseek`, `rewind` ili neki druga funkcija za promenu pozicije pozvana, svaki karakter koji se vraća nazad korišćenjem `ungetc` funkcije će biti odbačen.

Vraćanje karaktera u tok koji je na EOF poziciji, resetovaće EOF indikator za dati tok, jer nakon toga opet postoji karakter koji je dostupan za čitanje. Međutim, ukoliko je karakter EOF vraćen nazad u tok, `ungetc` ne radi ništa i samo vraća EOF.

Evo primera koda koji čita sve whitespace karaktere sa početka datoteke sa `getc`, onda vraća jedan bajt nazad kada preuzme prvi karakter koji nije razmak. Na kraju, čitaju se svi karakteri do newline karaktera korišćenjem `getline` funkcije.

```

#include <stdio.h>

int main()
{
    int in_char;
    FILE *my_stream;
    char *my_string = NULL;
    size_t nchars = 0;

    my_stream = fopen ("snazzyjazz.txt", "w");
    fprintf (my_stream, "          Here's some non-whitespace.\n");

    /* Close stream; skip error-checking for brevity of example */
    fclose (my_stream);

    my_stream = fopen ("snazzyjazz.txt", "r");

    /* Skip all whitespace in stream */
    do
        in_char = getc (my_stream);
    while (isspace (in_char));

    /* Back up to first non-whitespace character */
    ungetc (in_char, my_stream);

    getline (&my_string, &nchars, my_stream);

    /* Close stream; skip error-checking for brevity of example */
    fclose (my_stream);
}

```



```
printf ("String read:\n");
printf ("%s", my_string);

return 0;
}
```

Primer koda od gore će preskočiti sve whitespace karaktere iz datoteke snazzyjazz.txt i prikazati sledeći tekst na izlazu:

```
String read:
Here's some non-whitespace.
```

4. Programiranje korišćenjem cevi (pipes)

Postoje situacije kada je potrebno manipulirati drugim programima na GNU sistemu od strane korisničkom C programa. Jedan dobar način da se tako nešto uradi se naziva *ceva* (eng. pipe). Korišćenjem cevi, moguće je čitati ili upisivati u bilo koji program u okviru GNU sistema koji podrazumevano ispisuje rezultat na standardnom izlazu i čita podatke sa standardnog ulaza. Na prethodnicima modernih GNU sistema, cevi su bile često korišćene datoteke na disku; danas se obično koriste kao tokovi ili nešto slično. Nazivaju se cevi jer korisnici obično vizualizuju njihovo korišćenje kao podatke koji ulaze na jednom kraju, a izlaze na drugom.

Na primer, moguće je imati potrebu proslediti izlaz nekog programa ka printeru. Kao što je u uvodu rečeno, svaki printer u okviru sistema ima dodeljenu datoteku uređaja, kao što je /dev/lp0. Cevi omogućavaju bolji način da se prostledi izlaz ka printeru, u poređenju sa upisivanjem direktno u uređaj.

Cevi su korisne za mnoge stvari, ne samo slanje ispisa ka štampaču. Zamislimo situaciju u kojoj je potrebno izlistati sve programe i procese koji se izvršavaju na računaru koji sadrže string "init" u svom nazivu. Da bi se tako nešto uradilo na GNU/Linux komandnoj liniji, komanda je:

```
ps -A | grep init
```

Ova komanda prihvata izlaz ps -A komande, koja izlistava sve procese koji su aktivni, i prosleđuje putem cevi simbolom (|) komandi grep koja filtrira samo one redove koji sadrže string init. Izlaz bi bio, na primer:

```
1 ?      00:00:11 init
4884 tty6 00:00:00 xinit
```

Simbol za ceva | je veoma zgodan prilikom korišćenja cevi u okviru komandne linije ili shell skripti, ali je takođe moguće koristiti cevi i u okviru C programa. Dve osnovne C funkcije koje treba zapamtiti, u ovom kontekstu, su *popen* i *pclose*.

Popen funkcija prihvata kao prvi argument string koji sadrži shell komandu, kao na primer lpr. Drugi argument je string koji sadrži mod r ili w. Ako je specificiran r, ceva će biti otvorena za čitanje, a ako se specificira w, biće otvorena za pisanje. Povratna vrednost je tok otvoren za upis ili čitanje, u zavisnosti od toga kako se otvara ceva. Ako je došlo do greške, popen vraća null pokazivač.

Pclose funkcija zatvara ceva otvorenu sa popen. Prihvata jedan argument koji predstavlja tok koji treba zatvoriti. Čeka na tok da se zatvori i vraća status koda vraćenog od strane programa koji je pozvan putem popen funkcije.

Ukoliko se otvori ceva za čitanje ili pisanje, između popen i pclose poziva, moguće je čitati iz ili upisivati u ceva na isti način kao što bi se čitalo ili pisalo iz bilo kog drugog toka, korišćenjem ulazno/izlaznih rutina visokog nivoa, kao što su getdelim, fprintf i slično.

Program prikazan ispod pokazuje kako da se prosledi izlaz ps -A komande grep init komandi, na isti način kao što bi se to uradilo u okviru GNU/Linux komandne linije i primera pokazanog iznad. Izlaz programa bi trebao da bude identičan kao u primeru gore.

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

```

int
main ()
{
    FILE *ps_pipe;
    FILE *grep_pipe;

    int bytes_read;
    int nbytes = 100;
    char *my_string;

    /* Open our two pipes */
    ps_pipe = popen ("ps -A", "r");
    grep_pipe = popen ("grep init", "w");

    /* Check that pipes are non-null, therefore open */
    if ((!ps_pipe) || (!grep_pipe))
    {
        fprintf (stderr,
                "One or both pipes failed.\n");
        return EXIT_FAILURE;
    }

    /* Read from ps_pipe until two newlines */
    my_string = (char *) malloc (nbytes + 1);
    bytes_read = getdelim (&my_string, &nbytes, "\n\n", ps_pipe);

    /* Close ps_pipe, checking for errors */
    if (pclose (ps_pipe) != 0)
    {
        fprintf (stderr,
                "Could not run 'ps', or other error.");
    }

    /* Send output of 'ps -A' to 'grep init', with two newlines */
    fprintf (grep_pipe, "%s ", my_string);

    /* Close grep_pipe, checking for errors */
    if (pclose (grep_pipe) != 0)
    {
        fprintf (stderr,
                "Could not run 'grep', or other error.\n");
    }

    /* Exit! */
    return 0;
}

```