Nizi in besedilo

1 Uvod

Poleg dela s številkami od računalnika pogosto želimo, da nekaj naredi z nizi besedila. Primeri takšnih programov so npr. urejevalniki besedila, ki jih uporabljamo tako za pisanje "enostavnega" besedila (koda), kot tudi za razna obogatena besedila. Pravzaprav pa skoraj vsak računalniški program dela z besedilom; kadarkoli želimo uporabniku prikazati neke informacije, jih moramo namreč izpisati na zaslon. Ko smo delali s številkami, smo problem izpisovanja prepustili računalniku, ker je kodo za branje in izpisovanje številk k sreči napisal že nekdo drug. Za pisanje splošnih programov pa tovrstno znanje ne bo dovolj; zato si poglejmo osnove dela z besedili.
Pri slovenščini se naučimo, da je besedilo sestavljeno iz več odstavkov, odstavek iz več povedi, poved iz več stavkov, stavek iz več besed, besede pa iz več črk. Pri tem se moramo zavedati, da stavke ločimo z ločili (vejice, pike, klicaji, itd.), besede ločimo s presledki, posamične odstavke pa ločimo z zamikanjem prve povedi v desno. Za predstavitev v računalniku je tak model preveč zakompliciran, zato vzamemo bolj enostavnega. Besedila bomo predstavili z *nizi* (angl. *string*), ki bodo zaporedja več *znakov* (angl. *character*). Vse, kar bi si kadarkoli zaželeli izpisati, bomo proglasili za znak. Tako si bomo vse črke predstavljali kot znak, kjer bomo ločili tudi velikimi in malimi črkami (saj vendar izgledajo drugače, če jih napišemo), prav tako bomo za znake proglasili tudi ločila, oklepaje in matematične operacije (`+`, `-`, `*`, `/`). Poleg tega bomo za znak proglasili tudi števke od 0 do 9, ker tudi njih izpišemo.
Nenazadnje bomo ustvarili še nekaj posebnih znakov, ki jih morda nebi pričakovali. Od teh bomo zdaj spoznali tri: znak za presledek, znak za novo vrstico in znak za konec besedila. Znak za presledek bomo uporabili, kjerkoli želimo imeti prostor med dvema besedama (torej presledek). Za razliko od slovenščine tudi presledke obravnavamo, kot da bi bili pravzaprav neke posebne črke. Znak za novo vrstico bomo uporabili tam, kjer želimo, da izpis našega programa skoči vrstico nižje; brez tega znaka nam bo program vse izpisal v eni zelo dolgi vrstici besedila. Ta znak označimo s posebno kodo `\n` (ker se v angleščini ta znak imenuje *newline*), opazimo pa, da smo ga pravzaprav že srečali čisto na začetku. Uporabili bomo tudi znak za konec besedila, ki ga označimo z `\0`, pogosto pa mu rečemo tudi *NULL*. Več o temu znaku bomo povedali kasneje.

2 Predstavitev znakov

Preden si pogledamo nize, moramo razumeti, kako delamo z znaki. V C++-u imamo za to poseben tip `char`, ki nam hrani en znak. Če želimo spremenljivki tipa `char` nastaviti vrednost, moramo želeni znak dati v enojne narekovaje;

```
char crka = 'A';
```

Ta koda nam ustvari spremenljivko tipa `char`, ki hrani vrednost `'A'`, torej znak za veliko črko A. Tako kot števila lahko tudi znake pišemo in beremo; pri tem uporabimo `printf`, znotraj njega pa poseben formatnik `%c`, narejen za znake.

```
char crka;
scanf("%c", &crka);
printf("Tvoja crka: %c\n", crka);
```

Znak	ASCII koda
NULL (`\0`)	0
Nova vrstica (`\n`)	10
Presledek (`' '`)	32
`0`	48
`1`	49
`2`	50
⋮	⋮
`9`	57
`A`	65
`B`	66
`C`	67
⋮	⋮
`Z`	90
`a`	97
`b`	98
`c`	99
⋮	⋮
`z`	122

Tabela 1: Del ASCII tabele

Ker so računalniki narejeni za delo s številkami, moramo tudi znake predstaviti kot številke. To dosežemo s t.i. *kodnimi tabelami*, ki vsakemu znaku priredijo eno številko. Najpreprostejša kodna tabela je ASCII, ki lahko zakodira vse črke angleške abecede ter vse ostale zgoraj naštete znake, ne zmore pa zakodirati šumnikov. Le-teh se pri programiranju izogibamo, kar se le da. ASCII kode nekaterih pogostih znakov so prikazane v tabeli 1. Opazimo lahko, da so številke in črke v tabeli zaporedno; številka `0` ima kodo 48, številka `1` 49, ..., črka `A` ima kodo 65, `B` ima kodo 66, itd. Opazimo tudi, da so velike črke od malih ločene, in da imajo male črke večje kode.
Ta dejstva lahko uporabimo v programih tako, da črke preprosto obravnavamo, kot da bi bile številke. Črki `'a'` lahko npr. prištejemo neko številko, in tako dobimo črko, ki je toliko znakov naprej v abecedi; `'a' + 7` je na primer enako `'h'`. Poleg tega lahko znake med sabo primerjamo, kar bomo videli v prvem primeru.

Pogoste napake

ASCII koda je bila narejena v Ameriki specifično za angleško uporabo. Ker ne vsebuje šumnikov, le-teh ne moramo predstaviti v naših programih. Za delo s šumniki potrebujemo drugačne kodne tabele, ki jih tukaj ne bomo obravnavali.

Primer

Če poznamo kodne tabele, lahko na relativno enostaven način preverimo, če je neka črka velika ali majhna:

```
#include <stdio.h>

int main() {
    char crka;
    scanf("%c", &crka);
    if (crka >= 'A' && crka <= 'Z') {
        printf("To je velika crka\n");
    } else if (crka >= 'a' && crka <= 'z') {
        printf("To je majhna crka\n");
    } else {
        printf("To sploh ni crka!\n");
    }
    return 0;
}
```

Primer vhoda in izhoda

```
p
```

```
To je velika crka
```

Koda sprva prebere eno črko iz vhoda, nato pa preveri, če je vpisana črka med `A` in `Z`; če ni, potem preverimo še, če je črka med `a` in `z`.

Če dobro pogledamo v tabelo, vidimo, da koda 0 ne pripada števki `0`, pač pa znaku za konec besedila. To se morda na prvi pogled zdi nepričakovano, ampak ima svoj smisel; če je števka del besedila, o njej ne razmišljamo kot o številki, temveč pač o nekem znaku, ki ima v drugem kontekstu drugačen pomen. Če želimo to števko pretvoriti v številko, s katero lahko brez skrbi računamo, lahko uporabimo trik, kjer "odštejemo nič:"

```
char znak = '7';  // števka 7, napisana kot znak
int stevilka = znak - '0';  // če odštejemo nič, nič ne spremenimo

// NAROBE!
int to_pride_55 = znak - 0;
```

Pri tem triku moramo biti previdni, da odštejemo pravilno ničlo; če odštejemo številko `0`, se ne bo nič spremenilo; tako kot pri matematiki namreč odštevanje ničle številke ne spremeni. Če pa odštejemo *znak* `'0'` (v enojnih narekovajih), pa dejansko odštevamo številko 48, t.j. ASCII kodo znaka `'0'`. Praktično uporabo tega trika bomo pokazali v naslednjem delu.

3 Predstavitev nizov

Niz predstavimo kot zaporedje znakov. Ker o zaporedjih (oziroma natančneje o seznamih) nismo še ničesar povedali, moramo uvesti novo sintakso:

```
char niz_besedila[300];
```

Ta ukaz pove računalniku, naj ustvari spremenljivko z imenom **`niz_besedila`**, ki hrani *največ* 300 znakov. V tej spremenljivki bomo hranili naše zaporedje besedila. Če želimo nize brati ali pisati,

uporabimo funkciji, ki ju že poznamo, ter formatnik `%s`, tu pa je ena posebnost; za branje nizov ne napišemo znaka `&`. Če želimo neko spremenljivko nastaviti na niz, ki ga ne bomo prebrali, jo lahko nastavimo na običajen način z enačajem, ter z dvojnimi narekovaji. Tak način podajanja nizov smo že srečali; namreč vedno, ko uporabimo funkciji `scanf` ali `printf`.

Primer

Da se navadimo nove sintakse, si poglejmo preprost primer. Spodnji program prebere uporabnikovo ime in ga pozdravi.

```
#include <stdio.h>

int main() {
    char uporabnikovo_ime[300];
    scanf("%s", uporabnikovo_ime);  // NE napišemo &
    printf("Zivjo %s!\n", uporabnikovo_ime);
    return 0;
}
```

Ko ustvarimo niz, računalniku povemo, kolikšna je njegova najdaljša možna dolžina. Nič pa nam ne preprečuje, da v to spremenljivko shranimo krajši niz. Kako pa potem računalnik ve, kje se naš niz dejansko konča? Pričakujemo namreč, da bomo za zapis kratkega niza uporabili nekaj mest za znake na začetku, potem pa se bo niz nekje končal; kaj je na neuporabljenih mestih zaporedja, nas ne zanima. Ravno iz tega razloga so nizi zgrajeni tako, da imajo na koncu dodaten znak za konec besedila; znak NULL, ki smo ga omenili na začetku. Ta znak pove računalniku, da se besedilo tu konča in da naj naprej ne gleda. Če vrednost niza nastavimo z enačajem ali niz preberemo s `scanf`, bo računalnik sam poskrbel, da bo ta znak napisan na pravo mesto; če pa z nizi delamo kaj bolj zapletenega, moramo za ta znak skrbeti sami. Zaradi tega znaka je dejansko število vidnih znakov, ki jih lahko shranimo v niz, za eno manjše od predpisane največje dolžine. Da se tovrstnim problemom izognemo, bomo od sedaj naprej vedno napisali nekaj večjo število za dolžino niza; če pričakujemo, da uporabnik vpiše največ 200 znakov, bomo za velikost niza dejansko napisali 201 (ali celo malo več).
Preden pridemo do primerov, moramo spoznati še indeksiranje. Vsako mesto za znak v nizu ima svoj zaporedni *indeks*, s katerim lahko enolično dostopamo do tistega mesta. Indeksi so zaporedna števila, ki se začnejo z 0; prvi znak v nizu ima indeks 0, drugi znak ima indeks 1, tretji ima indeks 2, itd. Indeksiranje od 0 se morda sprva zdi nesmiselno, vendar je za tem zelo dober razlog, ki ga bomo spoznali pri obravnavi kazalcev (nadaljevalna skupina). Poglejmo si na primeru, kako deluje indeksiranje.

Znak	P	r	i	m	i	c	o	v	i	' '	J	u	l	j	i	NULL
Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

V zgornji tabeli je razpisan niz `"Primicovi Julji"`, pod vsakim znakom pa je napisan indeks, s katerim lahko dostopamo do tega mesta v nizu. Da zares dostopamo do teh znakov, uporabimo oglate oklepaje;

```
char niz[201] = "Primicovi Julji";
printf("Znak na prvem mestu: %c\n", niz[0]);  // P
niz[8] = 'a';
niz[14] = 'a';
printf("%s\n", niz);  // izpiše "Primicova Julja"
```

Ko zapišemo `niz[indeks]`, dobimo znak, ki ga lahko uporabljamo kot katerokoli drugo spremenljivko. Pri tem pazimo samo na to, da lahko niz spremenimo le, če ga spremenimo "direktno"; če znak prvo shranimo v neko drugo spremenljivko, in potem spremenimo to spremenljivko, se niz ne bo spremenil.

```
niz[3] = 'x';  // spremeni niz
```

```
char znak = niz[3];
znak = 'y';  // znak je sedaj 'y', niz pa je nespremenjen
```

Primer

Za prvi primer si poglejmo, kako bi lahko izračunali dolžino niza.

```
#include <stdio.h>

int main() {
    char niz[301];
    // uporabnik lahko napise niz dolžine najvec 300
    scanf("%s", niz);

    // da poiščemo dolžino, bomo dejansko poiskali indeks
    // znaka NULL; s tem bomo dobili točno število znakov pred njim
    int i = 0;  // trenutni indeks v nizu
    while (niz[i] != 0) {  // ASCII koda znaka NULL je 0
        i++;
    }
    // na kocnu je i ravno indeks znaka NULL, in s tem enak
    // dolzini niza
    printf("Niz je dolg %d znakov\n", i);
    return 0;
}
```

Ta koda ni težka, vendar jo je pogosto neprijetno pisati, zato imamo boljšo alternativo; če na začetek programa dodamo `#include <string.h>`, lahko uporabljamo funkcijo `strlen`, ki nam ravno tako izračuna dolžino niza:

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

int main() {
    char niz[201];
    scanf("%s", niz);
    printf("Dolzina niza: %d\n", strlen(niz));
    return 0;
}
```

Funkcijo `strlen` uporabljamo v skoraj vsakem programu z nizi, zato je dobro, da se je čim prej navadimo. Poleg tega `strlen` dolžino dejansko izračuna hitreje kakor zgornja koda.

Primer

V naslednjem primeru bomo napisali kodo, ki pretvori besedilo v velike črke. Za to uporabimo eno od lastnosti ASCII tabele, ki smo jo omenili prej; namreč, da so črke napisane zaporedno, da so velike črke pred malimi.

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

int main() {
    char niz[201];
    scanf("%s", niz);

    // Zamik med velikimi in majhnimi črkami je enak ne glede na to, katera
    // črka je to. V zanki bomo vsaki majhni črki odšteli zamik, s čimer jo
    // pretvorimo v veliko
    int zamik = 'a' - 'A';
    int dolzina = strlen(niz);
    for (int i = 0; i < dolzina; i++) {
        // če je črka majhna, jo moramo povečati
        // to moramo nujno preveriti, saj drugih znakov ne smemo spremeniti!
        if ('a' <= niz[i] && niz[i] <= 'z') {
            niz[i] = niz[i] - zamik;
        }
    }
    printf("%s\n", niz);
    return 0;
}
```

Primer

Za ta primer si poglejmo, kako bi pretvorili številko, zapisano z nizom, v številko, zapisano s številko. Prej omenjen trik z odštevanjem nič ne bo deloval, ker lahko z njim pretvarjamo le znake; lahko pa številko pretvorimo znak po znak. Pri tem pretvarjanju uporabljamo lastnosti desetiškega zapisa števil; namreč, da zaporedna mesta v zapisu predstavljajo vrednosti, ki se razlikujejo za faktor 10. Ko pretvorimo prvi del besedila, in želimo dopisati še eno številko, moramo že zapisani del "premakniti" eno mesto v levo, ter premaknjenemu številu prišteti novo številko. Premikanje dosežemo z množenjem z 10.

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

int main() {
    char niz[11];  // niz, ki bo hranil številko
    // premisli, zakaj je dolžina 11 že dovolj
    scanf("%s", niz);
    int dolzina = strlen(niz);
    int stevilo = 0;  // začnemo z 0
    for (int i = 0; i < dolzina; i++) {
        stevilo *= 10;
        stevilo += (niz[i] - '0');
        // niz[i] je znak, ki ga moramo pretvoriti v številko, da lahko
        // računamo z njim
    }
    printf("Stevilka je %d\n", stevilo);
    return 0;
}
```

4 Standardne funkcije

Izkaže se, da pri delu z nizi pogosto pišemo zelo podobne kose programa, kakor se je zgodilo pri primeru z izračunom dolžine. Namesto da večkrat napišemo skoraj enako kodo, so v knjižnici `string.h` dostopne razne funkcije, ki nam pogosto olajšajo delo.

4.1 Primerjava nizov

Za primerjavo dveh nizov ne uporabljamo dvojnega enačaja (`==`), temveč funkcijo `strcmp`. Funkcijo uporabimo tako, da ji v okrogle oklepaje napišemo dva niza; `strcmp(niz1, niz2)`. Če sta niza enaka, funkcija vrne rezultat `0`, sicer pa vrne drugačen rezultat. Spodnja koda preveri, če je uporabniku ime Filip:

```
char niz[101];
scanf("%s", niz);
if (strcmp(niz, "Filip") == 0) {
    printf("Ime ti je Filip\n");
} else {
    printf("Ni ti ime Filip\n");
}
```

Funkcija nam pravzaprav poda več informacij. Z njo lahko pogledamo, kakšna je *leksikografska ureditev* dveh nizov; preprosto povedano, kateri od nizov bi se, če bi bila oba niza besedi, pojavil prej v slovarju (leksikonu). Če bi se prvi niz pojavil prej, funkcija vrne negativno število. Če bi se drugi niz pojavil prej, pa funkcija vrne pozitivno število.

4.2 Kopiranje nizov

Če želimo eno spremenljivko prekopirati v drugo, lahko napišemo `b = a`. Na žalost pa to ne deluje za nize; namesto `niz2 = niz1` moramo napisati `strcpy(niz2, niz1)`. Funkcija `strcpy` prekopira drugi niz v prvega; na koncu bosta oba niza imela enako vsebino.
Če želimo nekemu nizu na konec dodati nek drug niz, lahko za to uporabimo funkcijo `strcat` (*string concatenate*). Ta funkcija prav tako sprejme dva niza; ko jo pokličemo, drug niz kopira na konec prvega.

4.3 Operacije na prvih n znakih

Včasih želimo kopirati ali primerjati le del niza. Za to imamo na voljo malce drugačne verzije zgoraj naštetih funkcij; če v imenih teh funkcij za `str` dodamo še `n` (torej `strncmp`, `strncpy`, ...), in funkciji kot zadnji argument podamo številko n, bo funkcija svoje delo opravila le na prvih n znakih; `strncmp(niz1, niz2, 3)` bo primerjal le prve tri znake, `strncpy(niz2, niz1, 7)` bo kopiral le prvih sedem znakov, ipd.

5 Napredno branje nizov

Pogosto se v nalogah pojavi, da moramo prebrati niz do konca vrstice, ali pa da sploh ne vemo, koliko vrstic vhoda bo do programa prišlo. Spopadanje s tovrstnimi problemi je opisano v zapiskih za nadaljevalno skupino iz tretje ure (Branje in pisanje 2).