5. PROCESIRANJE PODATAKA

Razlikujemo brojčane i znakovne, ili string podatke.

5.1 Matematičke operacije s Liberty BASICom

LB izvršava matematičke operacije na isti način kao i većina drugih programskih jezika.

5.1.1 Pravila pisanja i izračunavanja vrijednosti aritmetičkog izraza

Aritmetički izraz se sastoji od argumenata i aritmetičkih operacija. Simboli za aritmetičke operacije, odnosno aritmetički operatori u LBu dati su u tabeli 5.1, sa simbolima za sve operacije i stepene prioriteta kod njihovog izvršavanja.

Tabela 5.1 Aritmetičke operacije u LB jeziku

Aritmetička operacija	Simbol u LBu	Stepen prioriteta kod izvršavanja naredbe
Stepenovanje	^	1. (najviši)
Dijeljenje	/	2.
Množenje	*	2.
Oduzimanje	–	3.
Sabiranje	+	3.

Slijedeći primjer pokazuje da se aritmetičke operacije izvode slijeva nadesno, poštujući prioritet operacija:

```
v = 5 + 2 * 3          (Rezultat je: 11, jer se prvo izvodi operacija množenja)
```

Prioritet operacija se može narušiti korištenjem malih zagrada (). Izrazi u zagradama se prvi izračunavaju. Pri tome su unutrašnje zagrade višeg prioriteta od spoljašnjih. Primjer:

```
v = (5 + 2) * 3        (Rezultat je: x = 21)
```

Primjeri primjene operacije stepenovanja:

```
x = 10 ^ 3             (Rezultat je: x = 1000)
x = 81 ^ 0.5           (Rezultat je: x = 9, tj. korijen broja 81)
```

Primjer operacije dijeljenja:

```
d = 3 / 2              (Rezultat je: d = 1.5)
```

5.1.1 Funkcije

Funkcije su ugrađene rutine (potprogrami) koje manipulišu brojevima, stringovima i izlazima.

Iza imena funkcije uvijek idu male zagrade, a sadržina zagrade je argument funkcije.

5.1.2 Numeričke funkcije

Int()

Abs(numerička vrijednost)

Atn(numerička vrijednost) - daje arkus tangens argumenta u radijanima

2 ¶ rad = 360 °

1 rad = 180 / ¶ °

1 ° = ¶ / 180 rad

Sin(numerička vrijednost) - daje sinus ugla argumenta predstavljenog u radijanima

Cos(numerička vrijednost)

Tan(numerička vrijednost)

Exp(numerička vrijednost) - daje bazu prirodnog logaritma e stepenovanu sa numeričkom vrijednošću argumenta ($e^{argument}$)

Argument za funkciju Exp() može biti konstanta, promjenljiva, ili izraz.

e = 2.718282

Log(numerička vrijednost) - daje prirodni logaritam argumenta.

Argument funkcije Log može biti bilo koja konstanta, promjenljiva, ili izraz, čija vrijednost je pozitivna

5.2 Stringovi

Programski jezik BASIC i svi njegovi nasljednici, kao što su LB i Visual BASIC imaju odličnu podršku za stringove.

Može se kazati da BASIC i danas dominira, u odnosu na ostale programske jezike, u lakoći rada i manipulacije sa stringovima.

5.2.1 Spajanje stringova

Dva stringa mogu da se spoje u jedan jednostavnim postavljanjem znaka »+« između njih.

Primjer 5.1 Spajanje dva stringa

```
‘Filename: Merge.bas
Input "Unesi prezime:□"; Prezime$
Input "Unesi ime:□"; Ime$
PunoIme$ = Prezime$ + "□" + Ime$
Print "Prezime i ime:□"; PunoIme$
End
```

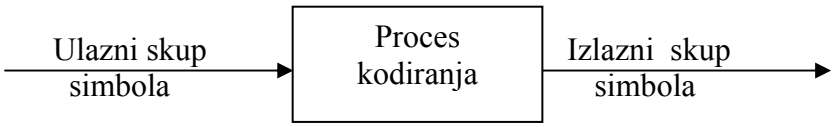
Program je uskladištio spojeni string u novu string-promjenljivu pod imenom `PunoIme$`. S ovom promjenljivom možemo raditi sve što želimo. Možemo na primjer štampati puno ime na koverti za dostavu podataka o obračunu plate radnika.

Postoji više vrsta brojčanih podataka, koje je u radu potrebno tretirati kao string-promjenljive:

- brojevi socijalnog osiguranja,
- poštanski brojevi,
- numerički kodovi inventara,
- brojevi dijelova sklopa, uređaja, ili mašine i dr.

5.2.2 Kodiranje i ASCII tabela

Očito je da se binarni brojevi lako mogu zapisati i obrađivati u računaru, jer su njegove komponente prilagođene ovakvom radu. Međutim, podaci i programi koje računaru dajemo na ulazu obično nisu u ovom obliku. Čovjeku je lakše sa računarom komunicirati preko standardnih znakova koje upotrebljava u svakodnevnom obavljanju svojih zadataka. To su razna slova, cifre decimalnog brojnog sistema, kao i neki specijalni znaci: (,), ?, !, = " itd. Da bi računar mogao da prihvati ove podatke, potrebno ih je kodirati. Kodiranje znači pretvaranje jednog skupa simbola u drugi skup i šematski je predstavljeno na slici 4.5.



Sl. 5.1 Pojednostavljena predstava procesa kodiranja

U našem slučaju ulazni skup simbola, koji se često zove i alfa-numerički skup znakova, prevodi se u niz "0" i "1" tako da ih računar može prihvatiti. Pri tome se kodiranje obavlja tako da se za svaki ulazni simbol odredi tačan niz "0" i "1" koji se zove kod datog simbola i koji se razlikuje od kodova svih drugih simbola.

Postavlja se pitanje koliko je binarnih cifara potrebno da bi se pomoću njihovih različitih kombinacija predstavio dati skup ulaznih znakova: sva slova, sve cifre i određeni specijalni znaci. Ako se uzme da postoji 10 cifara, oko 30 slova i oko 20 specijalnih znakova, zadatak je da se zakodira oko 60 ulaznih simbola. S druge strane, sa n binarnih cifara može se postići 2^n različitih stanja, tj. zapisati binarno 2^n različitih ulaznih simbola. Da bi se svi naši simboli mogli zapisati, potrebno je da bude $2^n \geq 60$. Iz ove nejednakosti slijedi da je najmanji broj n, koji je zadovoljava, jednak 6 ($2^6 = 64$).

Kodovi koji koriste 6 binarnih cifara za predstavljanje skupa alfa-numeričkih znakova danas se ne koriste i od njih su najpoznatiji BCD-kod i FIEL-DATA-kod.

Pri konstrukciji računara pokazalo se da je pogodno da ovaj kod bude nešto duži, da bi se pomoću njega mogao predstaviti znatno veći broj uglavnom specijalnih znakova, ali i da bi se u njega ugradile i određene kontrole. Zbog toga se danas najčešće koriste kodovi sa 8 binarnih cifara pomoću kojih se može predstaviti $2^8 = 256$ različtih znakova. Najpoznatiji iz ovog skupa kodova su: EBCDIC-kod (Extended binary-coded decimal) i ASCII-kod (American

Standard Code for Information Interchange – američki standardni kod za razmjenu informacija), koji je danas prevladao.

Primjeri iz ASCII koda:

Slovu A pridijeljena je vrijednost 65 (od 256), odnosno slovo A je kodirano sa 0 1 0 0 0 0 0 1 (65 u binarnom obliku), slovo B je kodirano sa 0 1 0 0 0 0 1 0 (66 u binarnom obliku). Dakle, u računaru se za slovo A pohranjuje njegov ASCII kod i to je jedini način da računar razlikuje različite znakove..

S druge strane, cifra 1 kodirana je sa 0 0 1 1 0 0 0 1 (redni broj 49 u ASCII tabeli),
dok cifra 5 ima kod 0 0 1 1 0 1 0 1 (redni broj 53).

Tačan raspored svih znakova obično se daje tabelarno.

Tabela 5.2 Dio tabele ASCII kodova

Redni broj u ASCII tabeli od 0 do 255	**Karakter**	**ASCII kod – redni broj u binarnom obliku**
32	Space	00100000
48	0	00110000
49	1	00110001
50	2	00110010
51	3	00110011
57	9	00111001
63	?	00111111
65	A	01000001
66	B	01000010
67	C	01000011
97	a	01100001
98	b	01100010

Ovi primjeri mogu poslužiti da bi se shvatila razlika između kodiranja i konverzije decimalnog u binarni broj. Kod slova i specijalnih znakova konverzija uopšte ne postoji, pa ne postoji ni problem razlikovanja.

Prilikom konverzije kompletan decimalni broj posmatra se kao jedinstven (a ne cifra po cifra) i konvertuje se u binarni oblik. U računaru se binarne operacije provode samo nad binarnim brojevima nastalim konverzijom, a ne i nad binarnim kodovima. Razliku sagledajmo i na slijedeća dva primjera:

decimalni broj 5
- kodiran ASCII: 0 0 1 1 0 1 0 1
- preveden u binarni oblik: 1 0 1

decimalni broj 15
- kodiran ASCII: 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 1
- preveden u binarni oblik: 1 1 1 1

Zbog ovih razlika između koda decimalnih brojeva i njihovog binarnog oblika, obično se usložnjava proces obrade numeričkih podataka u računaru. Naime, podaci se unose preko tastature u decimalnom obliku, računar vrši kodiranje i ovi podaci stiču zakodirani kao niz "0" i "1" u centralni dio. Međutim, oni se ne mogu odmah obrađivati, nego se prvo iz koda određenim algoritmima prevedu u odgovarajuću binarnu vrijednost. Tek tada se vrši obrada ovih podataka. Rezultate, čije su vrijednosti u binarnom obliku, treba ponovo prebaciti u kod i tek onda vratiti korisniku. Iako ovo izgleda dosta složeno, računar to obavlja brzo i efikasno, tako da korisnik to uopšte ne primjećuje.

Rezime

Kao što smo vidjeli, sadržaj nekog bajta može predstavljati više različitih informacija. U jednom bajtu vrijednost 65 može označavati numeričku cjelobrojnu vrijednost 65 ili veliko slovo A, ili komandu nekog višeg programskog jezika, ili komandu strojnog jezika. Postavlja se pitanje kako računar razlikuje značenje podatka u memoriji. Odgovor je: nikako. Ako u programu naredimo računaru da uzme sadržaj određenog bajta i promatra ga kao tekst, on će to i učiniti. Ako u istom tom programu računaru naredimo da sadržaj istog tog bajta promatra kao numeričku vrijednost, računar će to i učiniti. Prema tome, značenje pojedinog bajta zavisi od njegove primjene u datom trenutku.

5.2.3 String funkcije

String funkcije LBa rade na sličan način kao i numeričke funkcije. Kada im se proslijedi argument, one vraćaju vrijednost koja se može snimiti ili štampati. String funkcije omogućavaju i traženje pojedinačnih karaktera unutar stringa.

Primjenom ASCII tabele, može se odštampati bilo koji znak, koristeći se njegovim ASCII kodom. Ovo je posebno važno za znakove kojih nema na tastaturi. Ovi znakovi se mogu štampati sa funkcijom Chr$, koja se zove *karakter string funkcija*. Sintaksa ove naredbe je:

```
Chr$(redni broj znaka u ASCII tabeli)
```

Pored funkcije Chr$(), dvije dodatne funkcije takođe rade s ASCII tabelom:

```
Asc("string vrijednost")
Space$(numericka vrijednost)
```

Funkcija Asc() vraća ASCII redni broj znaka, koji joj je proslijeđen kao argument. Argument mora biti string od jednog ili više znakova. Funkcija Asc() uvijek vraća ASCII redni broj samo prvog znaka u stringu, bez obzira na njegovu dužinu. Na ovaj način može se dobiti ASCII redni broj prvog slova imena, ili prezimena, ako su pohranjeni u dvije promjenljive.

Primjer:

```
Print Asc("A"); "□"; Asc("B"); "□"; Asc("C")
```

daje rezultat:

```
65 66 67
```

Dobiveni su redni brojevi u ASCII tabeli za znakove A, B i C.

Primjer u kome se string promjenljive koriste kao argumenti:

```
slovo1$ = "A"
slovo2$ = "B"
slovo3$ = "C"
Print  Asc(slovo1$);  "□";  Asc(slovo2$);  "□";  Asc(slovo3$)
```

daje isti izlaz kao i prethodni primjer.

Primjer kada argument ima više od jednog znaka:

```
Print  Asc("Zenica")
```

štampa broj 90 (ASCII redni broj za veliko slovo Z).

Funkcija Space$() izdaje određeni broj praznih mjesta, definisan cjelobrojnim (integer) argumentom funkcije. Primjer:

```
Print  "X";  Space$(40);  "X"
```

Funkcija Len() se koristi kada se želi znati dužina stringa. Ona takođe radi s brojevima, da bi pokazala koliko memorije troše numeričke promjenljive.

Primjer kada funkcija Len() daje dužinu (broj znakova) string promjenljive, konstante ili izraza unutar zagrada:

```
Print  Len("abcdef")
```

U ovom primjeru izlaz je broj 6.

Ako se radi o praznom, odnosno null-stringu (nema ni jednog znaka), funkcija u izlazu daje vrijednost 0.

Funkcija Left$() zahtijeva dva argumenta, odvojena zarezom: string-promjenljivu ili izraz, iza koga slijedi cjelobrojna konstanta ili promjenljiva.

Primjer:

```
a$  =  "abcdefg"
Print  Left$(a$,1)
Print  Left$(a$,3)
Print  Left$(a$,7)
Print  Left$(a$,20)
```

daje slijedeći izlaz:

```
a
abc
abcdefg
abcdefg
```

Zadnji primjer pokazuje da ako se traži u izlazu više znakova nego što postoji u stringu, tada funkcija Left$() izdaje cijeli string i ne daje nikakvu poruku o grešci.

Funkcija Left$() radi na sličan način, tako što izdaje desne znakove stringa, prema slijedećem primjeru:

```
a$  =  "abcdefg"
Print  Right$(a$,1)
Print  Right$(a$,3)
Print  Right$(a$,7)
Print  Right$(a$,20)
```

daje slijedeći izlaz:

```
g
efg
abcdefg
abcdefg
```

Funkcija Mid$() izdvaja znakove iz stringa po određenom pravilu. Ova funkcija se koristi s tri argumenta. Prvi argument je sam string. Primjer:

```
a$ = "Basic FORTRAN COBOL C Pascal"
Print Mid$(a$,1,6)
Print Mid$(a$,7,7)
Print Mid$(a$,15,5)
Print Mid$(a$,21,1)
Print Mid$(a$,23,6)
```

Daje slijedeći izlaz:

```
Basic
FORTRAN
COBOL
C
Pascal
```

Funkcija Instr$() je funkcija za pretraživanje stringova. Primjer:

```
a$ = "Liberty Basic FORTRAN COBOL C Pascal"
Print Instr(a$, "FORTRAN")
Print Instr(a$, "COBOL")
Print Instr(a$, "C")
Print Instr(a$, "PL/I")
```

daje izlaz:

```
15
23
23
0
```