Elektrotehnički fakultet u Sarajevu

SEMINARSKI RAD

PROGRAMSKI JEZICI I PREVODIOCI

BASIC CALCULATOR – PROGRAMSKI JEZIK

Student: Ajla Mešić

Index: 17496

U Sarajevu, 15.12.2018

Uvod u jezik

Općenito

Bc je skraćeno od *basic calculator* (koji se još često naziva i *bench calculator*). Bc je kalkulator proizvoljne preciznosti (podržava brojeve proizvoljne preciznosti), čija je sintaksa slična C programskom jeziku. Obično se koristi kao matematički skriptni jezik ili kao interaktivna "matematička školjka"(mathematical shell). Dakle, može se koristiti interkativno, unošenjem instrukcija sa terminala, ili može pokrenuti programe iz nekih datoteka. Standardna matematička biblioteka dostupna je na komandnoj liniji. Ako se zatraži, matematička biblioteka se definiše prije obrade bilo koje datoteke. U bc jeziku ne postoji glavni program, već se kod izvršava onako kako sam jezik prolazi kroz njega.

Tipična interaktivna upotreba je kucanje komande bc na Unix komandnom promptu, gdje se unosi izraz kao što je (1+3)*2, nakon čega se vraća izlaz 8. Iako, kao što je već rečeno, bc može raditi da brojevima proizvoljne preciznosti, ako očekujemo da izlaz ulaza 2/3 bude određene preciznosti, bc zahtjeva unos 0 poslije decimalne tačke, u suprotnom izlaz je 0, što može iznenaditi nove korisnike koji nisu svjesni ove činjenice. Prihvataju se kako decimalni tako i nedecimalni brojevi. Oni koji su upoznati sa C jezikom, naći će nekoliko iznenađenja na ovom jeziku.

Ako se u komandoj liniji koriste datoteke, te datoteke moraju biti teksutalne koje sadrže bc instrukcije. Bc počinje sa obradom koda iz svih datoteka navedenih u komandoj liniji u navedenom redoslijedu. Nakon što su sve datoteke obrađene, izvršavaju se upute sa standardnog unosa. Svi kodovi se izvršavaju pri čitanju. Ako datoteka sadrži komandu za zaustavljanje procesora, bc nikada neće doći do standardnog ulaza tj. neće pročitati upute na standardnom ulazu. Bc završava svoj rad kada dodje do komande quit ili dođe do završnog fajla standarnog ulaza.

Dakle, izlaz kompajlera se tumači i izvršava zbirkom rutina koje rade sa ulazima, izlazima i općenito aritmetikom sa velikim brojevima. Rutine bc-a su zasnovane na dinamičkom raspoređivanju prostora za čuvanje podataka. Overflow se ne pojavljuje, sve dok prostor nije iscrpljen. Rutine su kolekcija koja se još naziva i dc.

Kompajler nije namijenjen da obezbijedi potpuni programski jezik. Bc je minimalni programski jezik.

Bc jezik ima potpunu strukturu kontrole kao i funkciju trenutnog rada (odmah izvršava instrukcije).

Funkcije se mogu definisati i sačuvati za kasnije izvršavanje. Na raspolaganju je i mala kolekcija bibliotečkih funkcija kao što su *sin, cos, arctan, log, exponential, Bessel funkcije integer redoslijeda.*

Neke od potreba za ovim jezikom su:

I. Računanje sa velikim brojevima
II. Rezultat treba biti precizan, na više decimala
III. Konverzija brojeva iz jedne u drugu bazu (brojevi se dakle mogu konvertovati iz npr decimalne u oktalnu bazu, jednostavno podešavajući izlaznu bazu na 8)

Manipluacija brojevima sa mnogo stotina cifara moguća je i na najmanjoj Unix verziji.

Kucanjem na komandnoj liniji bc -l, učitat će se skup funkcija biblioteke, koja se sastoji od već prije pomenutih funkcija. Biblioteka također postavlja scale na 20. Naknadno se može namjestiti na nešto drugo. Scale je opisan u nastavku.

Kucanjem na komandoj liniji bc -i, bc se stavlja u interkativni mod. U ovom režimu, bc prikazuje prompt kada očekuje neki unos. Pored toga, nešto drugačije se bavi greškama u ovom modu. Obično kada bc naiđe na grešku pri obradi datoteke, tumač prikazuje poruku o greški i izlazi, dok u ovom režimu tumač prikazuje poruku i vraća se u prompt da bi omogućio debugging.

Ako na komandnoj liniji upišemo bc file, bc učitava i izvršava datoteku nazvanu file prije prihvatanja komandi sa tastature kako je opisano prethodno. Na ovaj način se mogu učitati programi i definisane funkcije.

Ostale moguće opcije:

`-h, --help`
: Štampa pomoć i izlazi.

`-l, --mathlib`
: Definiše standardnu matematičku biblioteku.

`-w, --warn`
: Daje upozorenja za ekstenzije POSIX bc.

`-s, --standard`
: Procesira tačno POSIX bc jezik.

`-q, --quiet`
: Do not print the normal GNU bc welcome.

`-v, --version`
: Štampaj broj verzije i copyright i napusti.

Mogući rezultati izlaza:

0

: Successful completion.

1

: Desio se neki od sljedećih failova:

— break statement found outside of loop
— parser stack overflow
— syntax error
— end of file in comment
— end of file in string
— numerical constant is too long
— string is too long
— empty evaluation stack
— cannot pass scalar to array
— cannot pass array to scalar
— invalid array index
— built-in variable cannot be used as a parameter or auto variable
— *name* is not a function

— invalid value for built-in variable
— shell command failed to execute
— division by 0
— invalid value for exponentiation operator
— attempt to take square root of negative number
— out of memory
2

Unknown command line option.

Historija

Basic calculator se prvi put pojavio u Verziji 6 Unix 1975. godine. Napisali su ga Robert Morris i Lorinda Cherry iz Bell Labs-a. Bc je prethodio dc, raniji proizvoljni precizni kalkulator koji je napisan od strane istih autora. Dc je mogao izvoditi proizvoljne precizne proračune, ali njegova sintaksa sa obrnutom poljskom notacijom (RPN- *reverse Polish notation*) nije bila prilagođena korisnicima i zato je bc napisan kao prednji kraj dc-a (front-end). Bc je bio vrlo jednostavan kompajler (jedna YACC izvorna datoteka sa nekoliko stotina linija), koja je konvertovala novu C-like, bc sintaksu u dc postfiks notaciju i provodila rezultate kroz dc.

1991. godine POSIX je rigorozno definisao i standardizovao bc. Dvije implementacije ovog standarda i danas se koriste:

- Prva je tradicionalna Unix implementacija, prednja strana (front-end) na dc, koja se koristi u Unix I Plan 9 sistemima. Plan 9 bc je isti kao POSIX bc, osim što sadrži i ključnu riječ *print.*

```
print "The square of ", 2, "is ", 2*2
štampa se rezultat
The square of 2 is 4

print 1,2,3
štampa se rezultat
1 2 3

print 1,"",2,"",3
štampa se rezultat
123

print
štampa se rezultat


print 1,2,3, ; print 4, 5, 6
štampa se rezultat
1 2 3 4 5 6
```

- Druga je besplatni softver GNU bc, koji je prvi put objavio Philip A. Nelson 1991. godine. Implementacija GNU-a ima brojne nadogradnje izvan POSIX standarda i više nije front-end za dc (to je bytecode interpreter). Pisan je u C-u. U nastavku je detaljnije objašnjen.

Opis sintakse i semantike

Sintaksa u BNF ili EBNF nije pronađena.

Gramatika

Imena, vezanja i opseg

Identifikatori se koriste kao imena za varijable, funkcije i nizove. Mogućnost korištenja identifikatora više od jednog karatera u dužini je ekstenzija koja se ne nalazi u tradicionalnim implementacijama bc-a(kod POSIX bc dozvoljen je samo jedan karakter za ime). Inače, validni identifikatori mogu da sadrže sekvence u kojima može biti bilo koji broj slova,cifara ili znak (_), ali moraju početi malim slovom. Prazni prostori (spaces) nisu dozvoljeni u identifikatorima.

- Varijabla sadrži jednu numeričku vrijednost. Mogu se deklarirati kao lokalne za funkciju pomoću *auto* izraza. Sve ostale varijable su globalne i mogu se koristiti bilo gdje. Globalne varijable se ne moraju deklarisati. Bc kreira varijable kako ih zahtijeva, čija je početna vrijednost 0. Postoji i specijalna varijabla . (dot) koja sadrži rezultat posljednjeg izračunavanja).
- Funkcija je niz instrukcija koje izračunavaju jednu vrijednost. Lista nula ili više vrijednosti priloženih u zagradama uvijek prati naziv funkcije kao npr `moja_fja (3.14159)`. Funkcije se definišu koristeći ključnu riječ *define.* Kao i obično, postoji *return* koji je praćen povratnom vrijednosti u zagradama. Ako nema definisanu povratnu vrijednost ili nema izvještaja o povratku vraća 0. Funkcije, koje se kasnije razmatraju, definišu se kada se susretnu.
- Niz je lista vrijednosti. Vrijednosti na listi se nazivaju elementi niza. Svaki element u nizu je numerisan(indeksiran), počevši od 0. Takav broj se naziva indeks. Naprimjer, `a[0]` se odnosi na prvi element iz niza a. Ako je indeksna vrijednost tipa float, frakcijski dio se odbaci da bi se indeks pretvorio u integer. Naprimjer, sljedeći izrazi se odnose na isti element: `a[3] a[3.2] a[3.999]`. Ovi izrazi se mogu slobodno koristiti u izrazima, funkcijama i sl. Za razliku od mnogih jezika, veličina niza u bc se ne mora deklarisati. Elementi se kreiraju dinamički po potrebi, sa početnom vrijednosti 0. U bc dozvoljeni su samo *jednodimenzionalni nizovi.* Kada se koristi samo ime niza za neku funkciju, tada se čitav niz kopira za njegovo korištenje u datoj funkciji, gdje se nakon završetka funkcije odbacuje. Samo ime niza u nekom drugom slučaju/kontekstu se ne može koristiti.

Pošto zagrade uvijek prate imena funkcija i uglaste zagrade uvijek prate imena nizova, *bc može razlikovati 3 tipa imena*. Zbog toga može postojati varijabla, funkcija i niz sa istim imenom.

Naprimjer:

I. ○ kao varijabla
II. ○ `( )` kao funkcija
III. ○ `[ ]` kao niz

- Najosnovniji element ovog jezika je broj. Brojevi su proizvoljne preciznosti. Preciznost se odnosi na integer dio, kao i na frakcioni dio. Svi brojevi su interno prikazani u decimalnom broju i sva računanja se vrše u decimalnom broju. Bc pretvara konstante u interne decimalne brojeve, koristeći trenutnu ulaznu bazu (ibase). Postoje dva atributa brojeva: *length i scale*. Length je ukupan broj značajnih decimalnih cifara u broju, a scale stoji za ukupan broj decimalnih cifara nakon decimalne tačke. Vrijednost atributa scale je do 99, a po defaultu je 0. Mogu da sadrže opcionalni znak – (minus), dok . označava decimalni broj. Validne cifre su od 0 do 9 i heksadecimalne cifre od A do F. Velika slova su vrijednosti od 10 do 15. Mora biti najmanje jedna cifra prije ili poslije decimalne tačke(.). Ako nije, bc interpretira decimalnu tačku kao posebnu varijablu. Broj može biti proizvoljno dug i može sadržavati razmake. Nekoliko validnih brojeva sa ulaznom bazom 10: 0 0. .0 -3.14159 +09. -12 1 000 000. Nekoliko validnih brojeva sa bazom 16(ibase=16): 0 FF FF.3 -10.444 A1. Na kraju treba se dodati, da je validno i 1000000 i 1 000 000, ali ne i 1,000,000 čime ovo rezultira greškom.

Pošto postoji samo jedna vrsta broja, *ne postoje pravila za miješanje tipova*. Umjesto toga, postoje pravila o skali izraza, koja će biti objašnjena u narednim poglavljima.

Pored svega navedenog, sadrži i poznate kontrukcije kao što su if(cond); while(cond); for(init;cond;inc) iz C. Detaljnije u nastavku.

Baza ulaznih podataka(u interkativnom režimu), izlaznih kao i programskih konstati, mogu se specificirati postavljanjem rezervisane riječi *ibase* (input base) i *obase* (output base). Validne vrijednosti ibase su od 2 do 16. Nevalidna dodjela izvan ovog opsega rezultirat će vrijednošću 2 ili 16.

Rezervisane riječi:

I. *ibase*
II. *if*
III. *obase*
IV. *break*
V. *scale*
VI. *define*
VII. *sqrt*
VIII. *auto*
IX. *length*
X. *return*
XI. *while*
XII. *quit*
XIII. *for*

Za razliku od C-a, prilikom ulaska u funkciju sve stare vrijednosti parametara funkcije bivaju poslane na stek. Dok se ne izvrši return u funkciji, referenca na parametre se odnosi samo na nove vrijednosti.

Tipovi podataka, izrazi i programske strukture

Najjednostavniji izraz je aritmetički izraz na liniji po sebi. Kao naprimjer, ako upišemo na liniji:

142857 + 285714, program će odmah da odgovori sa linijom 428571.

- Operacije koje se još mogu naći pored sabiranja su − (oduzimanje), *(množenje), /(dijeljenje), %(modul), i ^ (stepenovanje). Dijeljenje integera daje integer, a dijeljenje sa 0 rezultira komentarom greške.

Svaki broj u izrazu može biti predefinisan znakom minus da se označi njegova negativnost (odnosi se na unarni znak minus -). Izraz 7+ -3 znači da broj -3 treba dodati na 7.

U kompleksnijim izrazima sa nekoliko operatora i zagradama postoje prioriteti nad operacijama.

- Najveći prioritet ima ^ operator, zatim * i % i /, pa poslije toga + i -. U izrazu prvo se gleda ono što je u zagradi, to se izračunava, a nakon toga ono što je van zagrada. Stepenovanje se izvodi sa desna na lijevo, dok se drugi operatori izvode sa lijeva na desno.

Dva sljedeća izraza su ekvivalentna:

```
I.   a^b^c i a^(b^c)
II.  a*b*c i (a*b)*c
III. a/b*c i (a/b)*c
```

Unutrašnji registri za čuvanje brojeva su označeni malim slovima. Vrijednost izraza može se dodijeliti registru na uobičajen način. Izraz x = x+3 označava povećanja vrijednosti iz x za 3. Kada je krajnji operator = kao u ovom slučaju, zadatak se vrši, a rezultat se ne štampa.

Neke napomene vezane za operacije među izrazima (pravila *scale*):

I. `expression ^ expression` – drugi izraz (expression) mora biti integer. Ako je *scale* prvog izraza a , a drugog b, tada je *scale* rezultata:
   min (axb, max (*scale*, a)).
II. `expression * expression` – ako su a i b *scale* ova dva izraza, *scale* rezultata je
   min (a+b, max (scale,a,b))
III. `expression / expression` – *scale* rezultata je vrijednost ***scale***.
IV. `expression % expression` – isto što i a-a/b*b. *Scale* rezultata je suma scale *divisor*-a i vrijednost ***scale***.
V. `expression + expression` – *scale* rezultata je maksimum između scale izraza.
VI. `expression - expression` – *scale* rezultata je maksimum između scale izraza.

Primjer operacije modula (%) za različite vrijednosti *scale*:

- `scale=0; 5%3`

  štampa se rezultat

  2

- `scale=1; 5%3`

  štampa se rezultat

  .2

- `scale=20; 5%3`

  štampa se rezultat

  .00000000000000000002

Po defaultu, bc prikazuje rezultat bilo koje instrukcije na novu liniju. Također čuva posljednju izračunatu vrijednost u specijalnu varijablu, koja je spomenuta ranije (.), tako da je moguće vršiti kalkulacije, što je prikazano u primjeru ispod.

- `2 + 2`

  štampa se 4

  `.*10`

  štampa se 40

Ibase, obase

Pošto je već u nekom od prethodnih poglavlja spomenuto značenje ibase i obase, u nastavku je opisano kako se njihove vrijednosti mijenjaju:

```
➢ ibase = 8

  11

  štampa se rezultat

  9
```

Ovim sada možemo da radimo sa oktalnom bazom. Međutim, ako želimo da vratimo u decimalnu bazu moramo paziti zato što

```
➢ ibase = 10
```

neće imati nikakav efekat, jer je broj 10 interpretiran u oktalnoj bazi. Umjesto toga, sjetimo se heksadecimalnih notacija A-F, tako da umjesto 10 koristit ćemo A.

Stoga, da vratimo u decimalnu bazu, pišemo

```
➢ ibase = A
```

Izlazna baza (obase) kao i ulazna (ibase), po defaultu ima vrijednost 10 i koristi se za izlazne brojeve (rezultate). Primjer ispod mijenja izlaznu bazu u heksadecimalnu:

```
➢ obase = 16

  1000

  štampa se rezultat

  3E8
```

što za rezultat daje trocifreni heksadecimalni broj. Dozvoljene su i velike izlazne baze i ponekad su korisne. Naprimjer, veliki brojevi mogu biti na izlazu u grupama od po 5 cifara postavljanjem obase do 100000. Neočekivane vrijednosti baza (1, 0 ili negativne) su kontrolisane.

Primjer za prethodno opisano dato je u nastavku:

```
➢ obase=1000

  123456789

  štampa se rezultat 123 456 789
```

Veoma veliki brojevi su podijeljeni na linije sa 70 znakova po liniji. Linije koje se nastavljaju, završavaju sa \. Pretvorba decimalnog izlaza se dešava odmah, dok izlaz velikih brojeva(više od 100 cifara) sa drugim bazama se dešava dosta sporije(3s).

- Zaključak je da ibase/obase utiču samo na koverziju ulaza/izlaza respektivno.
- *Scale, ibase i obase* se mogu koristiti kao i obične varijable.

- `scale = scale + 1,`

povećava vrijednost scale za 1, a biva štampana trenutna vrijednost scale. Iako se koristi kao varijabla*, zadržava svoje značenje kao broj decimalnih cifara iza tačke.*

- *Scale* je po defaultu 0, a ako je *bc -l* specificirano, kao što je već rečeno tada je `scale = 20`.

Niz (array)

U nekom od prethodnih poglavlja je objašnjeno šta je i kako funkcioniše niz.

Niz se može koristiti kao argument za funkciju ili se može deklarisati automatski u funkciji korištenjem praznih zagrada:

```
 I.   f(a[ ])
II.   define f(a[ ])
III.  auto a[ ]
```

Ostale bitne stvari su pomenute ranije.

Kontrolni izrazi

If, while i for se mogu koristiti za promjenu toka programa ili za stvaranje iteracije. Opseg svake od njih je izraz ili složeni izraz koji se sastoji od više izraza i nalazi u zagradama.

```
 I.   if (relation) statement
II.   while (relation) statement
III.  for (expression1; relation; expression2) statement
```

ili

```
 I.   if (relation) {statements}
II.   while (relation) {statements}
III.  for (expression1; relation; expression2) {statements}
```

- Za razliku od C-a, npr kod for petlje moraju se navesti sva tri izraza, nijedan se ne smije izostaviti.
- Relation stoji za neki izraz kao npr x > y, gdje pored > imamo i sljedeće operande za poređenje: <, <=, >=, ==(jednako) i !=(nije jednako).

To bi izgledalo ovako:

```
 I.   expression < expression
II.   expression > expression
III.  expression <= expression
IV.   expression >= expression
 V.   expression == expression
VI.   expression != expression
```

Treba paziti na razliku između = i ==. Nažalost, oba operanda su legalna, tako da neće biti prikazana nikakva greška, ali = nije operand za poređenje.

- `If` će se izvršiti samo ako je relation tačan (true). Onda if dalje prolazi kroz statements.
- `While` će se ponovno izvršiti ako je relation tačan (true). Relation se testira svaki put prije izvršenja tijela petlje (statements) i ako je relation netačan (false) while se prekida.
- `For` počinje izvršavanjem expression1, gdje nakon toga se provjerava relation i ako je istina izvodi se tijelo petlje (statements). Poslije toga se izvršava expression2 i tako sve dok relation ne bude false.

```
➢ for(i=1; i<=10; i=i+1)
  štampa integere od 1 do 10.
```

- Prisutan je i `break` za prekidanje `for` i `while` petlje.

Neke stvari koje je korisno znati:

- Sljedeća linija prepisuje vrijednost izraza sa desne strane u x i također povećava i prije nego što se on koristi kao indeks: `x = a[i=i+1]`
- *`statement ; statement ..`* – sekvenca statement-a, gdje je ; ekvivalent novoj liniji. Izvršava se s lijeva na desno.
- *{*
  statement
  statement
  …
  } – grupiše se više statement-a i ovo se najčešće viđa kod if i while.

```
➢ "any string"
  štampa string unutar navodnika
  any string
```

To nam dozvoljava da uradimo sljedeće:

```
➢ o = 15
  "The value of foo is "; o
  štampa se rezultat
  The value of o is 15
```

Izračunavanje pi na 1000 mjesta pomoću *arctangent* funkcije:

```
➢ $ bc -lq
  scale=10000
  4*a(1) # tan od 1 je 45 stepeni, što je pi/4 u radijanima.
  # Ovo može da potraje  nekoliko minuta.
```

Ako želimo *interaktivno* koristiti bc, to možemo pomoću *shell skripti*:

```
➢ $ result=$(echo "scale=2; 5 * 7 /3;" | bc)
  $ echo $result
  11.66
```

bash shell vraća samo integer

```
➢ $ result=$((5 * 7 /3))
  $ echo $result
  11
```

string *(u bash, ksh, csh)*

```
$ bc -l <<< "5*7/3"

11.66666666666666666
```

Sljedeće kontrukcije rade u bc-u na isti način kao i u C jeziku:

```
 I.   x=y=z  je isto kao x=(y=z)
II.   x=+y               x=x+y
III.  x=-y               x=x-y
IV.   x=*y               x=x*y
 V.   x=/y               x=x/y
VI.   x=%y               x=x%y
VII.  x=^y               x=x^y
VIII. x++                x=x+1
IX.   x--                x=x-1
 X.   ++x               (x = x+1)-1
XI.   --x               (x = x-1)+1
```

Neke od ugrađenih funkcija su:

- **sqrt** (expression) – rezultat je kvadratni korijen izraza (expression). *Scale* rezultata je veća vrijednost *scale* expression-a ili vrijednost *scale*, u zavisnosti koja je veća.
  Rezultat je skraćen na cijeli broj. Ovo je jedina matematička funkcija koja je ugrađena. Ostale su dostupne u vanjskoj standardnoj biblioteci.

```
x = sqrt(191)

x

štampa se rezultat

13
```

- **length** (expression) – rezultat je broj decimala izraza (expression). *Scale* rezultata je 0. Rezultat je uvijek integer.

- **scale** (expression) – rezultat je broj decimala iza tačke izraza (expression), tj *scale* izraza. *Scale* rezultata je 0. Rezultat je uvijek integer.

U nastavku primjeri za pojašnjenje rada length i scale.

- `length(.000001)`

  `štampa se rezultat`

  `6`

- `scale(1935.000)`

  `štampa rezultat`

  `3`

Mogu se dodati i komentari kao i u C-u */* komentar */.*

Standardna biblioteka funkcija (definirana sa -l opcijom u komandoj liniji)

bc komanda	funkcija	opis
s(x)	Sine	argument x – ugao u radijanima
c(x)	Cosine	argument x – ugao u radijanima
a(x)	Arctangent	vraća radijane
l(x)	Natural logarithm	
e(x)	Exponential function	
j(n,x)	Bessel function	vraća red n Besselove funkcije od x

Neki od prioriteta su sljedeći (najniži do najviši):

```
 I.   || operator, left associative
II.   && operator, left associative
III.  ! operator, nonassociative
IV.   Relational operators, left associative
V.    Assignment operator, right associative
VI.   + and - operators, left associative
VII.  *, / and % operators, left associative
VIII. ^ operator, right associative
IX.   unary - operator, nonassociative
X.    ++ and -- operators, nonassociative
```

Primjer rada jedne linije na osnovu prioriteta:

- `a = 3 < 5,`

bc prvo dodjeljuje varijabli a broj 3, zatim poredi 3 sa 5. Najbolje je koristiti zagrade.

Ako govorimo o **GNU bc-u** postoje neka znantna poboljšanja kao što su:

I. GNU bc varijable, nizovi i imena funkcija mogu sadržavati više od jednog karaktera.
II. Neki operatori iz C-a su uključeni.
III. Također if klauza može biti popraćena else klauzom.
IV. Omogućeni su i single line kometari sa #.
V. Posljednji rezultat se uvijek čuva, bez obzira da li je sačuvan u neku varijablu ili ne.
VI. Maksimalni broj elemenata u nizu je data od strane konfiguracijske varijable `{BC_DIM_MAX}`. Validni indeksi su od `0 do {BC_DIM_MAX}-1` (uključujući). Prijašnje ograničenje je bilo do 2048 elemenata.
VII. Također, prethodno vrijedi kako za scale tako i za obase. Opisano u nastavku.
VIII. Omogućene void funkcije.
IX. String ne može sadržavati nul znakove.
X. Postoji print
XI. Postoji continue

`BC_BASE_MAX` obase je trenutno 999. Maximum ibase je 16.
`BC_DIM_MAX` trenutni limit za niz je 65535 .
`BC_SCALE_MAX` brojevi prije i poslije decimale su ograničeni na maximum integera.
`BC_STRING_MAX` također ograničeni na int_max.
`exponent` ograničen na long_max
`variable names` trenutno ograničenje je 32767 posebno za niz, varijablu, funkciju.

- Pored logičkih operatora u POSIX bc-u, neka novododana iz C-a su:

  `&&` `||` `!`

Zagrade koje zatvaraju klauze moraju biti tačno jedna ispod druge kako je dato u primjeru ispod.

```
if (a<10) {
"a "
"is "; "less than 10 " a
}
else {
"a is"
" greater than 10 " a
}
```

ili alternativa

```
if(expression){statement1}\

else{ statement2}
```

Potprogrami

Funkcije

Ime funkcije je jedno malo slovo (POSIX bc). Dozvoljeno je definisanje 26 funkcija, pored 26 varijabli sa istim imenom.

```
define a(x){
```

predstavlja početak definisanja funkcije sa jednim argumentom x. Na dalje, mora slijediti tijelo funkcije nakon čega se funkcija zatvara sa }. Return se javlja kada se izvrši povratna vrijednost ili kada se dostigne kraj funkcije.

Postoje dvije forme:

```
 I.   return
II.   return(x)
```

- U prvom slučaju, povratna vrijednost je 0, a u drugom je vrijednost sadržana u x (tj. izraz koji se nalazi u zagradi).

- Varijable u funkciji se deklarišu naredbom `auto x, y, z`
  gdje može postojati samo jedna 'auto' naredba u funkciji i onda mora biti prva u tijelu funkcije. Varijable koje su na ovaj način deklarisane su alocirane i inicijalizirane na 0. Nakon izvršenja return-a one se odbacuju. Bilo koja varijabla sa istim imenom van funkcije, nije poremećena.

- Funkcije se mogu pozivati *rekurzivno*, ali automatske varijable su zaštićene na svakom nivou poziva.

Parametri navedeni u definiciji funkcije, tretiraju se na isti način kao i automatske varijable, sa jednom razlikom - oni dobijaju vrijednost prilikom ulaska u funkciju.

```
define a(x,y){
auto z
z = x*y
return (z)
}
```

- Funkcija se poziva kada se nailazi na njeno ime, a iza imena slijedi string argumenata u zagradama, koji su odvojeni zarezom (,). Rezultat je nepredvidiv, iz razloga što se ne zna šta se šalje kao argument (tj. može se poslati nevalidni argument za tu funkciju). Za poziv funkcija bez argumenata jednostavno pišemo njeno ime bez navođenja argumenata npr b( ).

```
a(7, 3.14)

štampa se rezultat

21.98

x = a(a(3,4), 5)

štampa se rezultat

60
```

Ostali primjeri

```
define f(n){
      auto i, x
      x=1
      for(i=1; i<=n; i=i+1) x=x*i
      return(x)
}

f(a)
štampa faktorijel od a, ako je a pozitivni integer.
```

Funkcija binomnog koeficijenta (m i n pozitivni integeri)

```
define b(n,m){

auto x, j

x=1

for(j=1; j<=m; j=j+1) x=x*(n-j+1)/j

return(x)

}
```

Funkcija koja izračunava vrijednosti eksponencijalne funkcije sabiranjem odgovarajućih serija bez obzira na eventualne greške skraćivanja:

```
scale = 20
define e(x){
auto a, b, c, d, n
a=1
b=1
c=1
d=0
n=1
while(1==1){
a = a*x
b = b*n
c=c+a/b
n=n+1
if(c==d) return(c)
d=c
}
}
```

Funkcija stepenovanja (s obzirom, kako je već prethodno rečeno, bc ne dozvoljava da stepen bude decimalni broj, pa korisnici možda prvo što naprave bi bila ova funkcija):

```
/*Funkcija koja vraća integer argumenta x*/

define i(x) {
auto s
s = scale
scale = 0
x /= 1
scale = s
return (x)
}

/* Iskorištena činjenica  da je x^y == e^(y*log(x)) */

define p(x,y) {
if (y == i(y)) {
return (x ^ y)
}
return ( e( y * l(x) ) )
}
```

GNU bc

Funkcija iz C-a u GNU bc:

C

```
double Phi(double x)
{
long double s=x,t=0,b=x,q=x*x,i=1;
while(s!=t)
s=(t=s)+(b*=q/(i+=2));
return .5+s*exp(-.5*q-.91893853320467274178L);
}
```

GNU bc

```
define phi(x) {
auto s,t,b,q,i,const
s=x; t=0; b=x; q=x*x; i=1
while(s!=t)
s=(t=s)+(b*=q/(i+=2))
const=0.5*l(8*a(1))   # 0.91893...
return .5+s*e(-.5*q-const)
}
```

Funkcija za računanje takse koja se šalje u procentima, gdje purchase predstavlja iznos kupovine:

```
define sales_tax(purchase,tax) {
auto old_scale
old_scale = scale; scale = 2
tax = purchase*(tax/100)
scale = old_scale
return (tax)
}

sales_tax(23.99,6)

štampa se rezultat

1.43
```

Neke decimale se mogu izgubiti tako da je bolje scale postaviti na veću vrijednost od 2.

Pri dijeljenju scale se mijenja. Ako želimo da je sačuvamo možemo napraviti sljedeću funkciju:

```
define integer_part(x) {
# lokalna varijabla
auto old_scale
# spasi staru scale, i onda je postavi na 0
old_scale = scale; scale = 0
# da dobijemo integer
x /= 1
# vrati staru skalu
scale = old_scale
return (x)
}
```

Sa ovako definisanom funkcijom, sada možemo napraviti funkciju za frakcionalni dio:

```
define fractional_part(x) {
return (x - integer_part(x))
}
```

Sljedeća funkcija omogućava da odaberemo scale i postavimo je za željeni broj:

```
define set_scale(x, s) {
auto os
os = scale
scale = s
x /= 1
scale = os
return (x)
}
```

`set_scale()` se sada može koristiti za sljedeću funkciju koja zaokružuje broj na dvije decimale :

```
define round2(num) {
auto temp;
if(scale(num) < 2) return (set_scale(num, 2))
temp = (num - set_scale(num, 2)) * 1000
if(temp > 5) num += 0.01
return (set_scale(num,2))
}
```

Sada se prvobitna funkcija može modifikovati, tako što ćemo koristiti prethodnu funkciju koja zaokružuje broj na dvije decimale:

```
define sales_tax(purchase,tax) {
auto old_scale
old_scale = scale
scale = 2
tax = round2(purchase*(tax/100))
scale = old_scale
return (tax)
}
```

Rekurzivna funkcija za Fibbonacijeve brojeve

```
define fib(n) {
if(n < 3) {
return (1)
}
else {
return (fib(n-1)+fib(n-2))
}
}
```

Bc koristi dinamičku alokaciju.

```
a=10

define f1() {
auto a;
a = 13;
return (f2())
}

define f2() {
return (a)
}


f1()

štampa se rezultat

13

f2()

štampa se rezultat

10
```

Razlog ovih rezultata je sljedeći:

- Kada se poziva f1( ), sakriva staru globalnu varijablu a i stvara novu lokalnu varijablu a i stavlja je na 13. Nakon returna lokalna varijabla se odbacuje.
- Kada se poziva f2( ) u njoj se ne stvara nova lokalna varijabla a, tako da jedina varijabla a koju ona vidi je globalna varijabla čija je vrijednost 10.

Apstraktni tipovi podataka i enkapsulacija

U prethodnim poglavljima smo vidjeli sve mogućnosti unutar bc jezika (od deklarisanja varijabli, nizova, do definisanja funkcija kao i njihova upotreba). Tako da se može zaključiti da je apstrakcija jedino podržana kroz smisao samih funkcija, ako to možemo tako opisati, pa zaključujemo da bc jezik nije objektno orijentisani jezik, a samim tim ne podržava apstraktne tipove podataka tj, ne podržava kreiranje, definisanje i korištenje tipova podataka nastalih od strane korisnika.

Podrška OOP

Kao što je već rečeno i objašnjeno, bc jezik ne podržava objektno-orijentirano programiranje.

Konkurentnost i obrada izuzetaka

Bc jezik ne podržava konkurentnost. On izvršava naredbe kao liniju po liniju. Dok se izvršavanje jedne linije ne završi, ne prelazi se na drugu. Ako su zadate skripte, bc prije komandi na komandnoj liniji, vrši zadatke iz skripti.

Što se tiče obrade izuzetaka, odnosno errora bc ne daje tu mogućnost osim što informiše o kojoj se greški radi. 0 označava da je zadatak uspješno završen, dok ostali brojevi govore o kakvoj greški je riječ.

Bc jezik ne podržava upravljanje događajima.

Mogući rezultati izlaza:

0

Successful completion.

1

Desio se neki od sljedećih errora:

— break statement found outside of loop
— parser stack overflow
— syntax error
— end of file in comment
— end of file in string
— numerical constant is too long
— string is too long
— empty evaluation stack
— cannot pass scalar to array
— cannot pass array to scalar
— invalid array index
— built-in variable cannot be used as a parameter or auto variable
— *name* is not a function
— invalid value for built-in variable
— shell command failed to execute
— division by 0
— invalid value for exponentiation operator

— attempt to take square root of negative number
— out of memory

2

Unknown command line option.

Funkcionalni programski jezici

Bc jezik ne podržava lambda funkcije niti operacije nad funkcijama. Kako je bc proceduralni jezik, on se ne može nazvati čistim funkcionalnim jezikom, ali mnoga svojstva funkcionalnog programiranja su implementirana unutar bc-a. Dakle, podržava, kao što smo već vidjeli, petlje, varijable koje mijenjaju stanje, definisanje funkcija i sl.

Bc jezik podržava lijenu evaluaciju.

Logički programski jezici

Kako bc nije deklarativan programski jezik, isti nije moguće napisati isključivo od deklaracija.

Reference:

http://www.unixprogram.com/bc.pdf

https://en.wikipedia.org/wiki/Bc_(programming_language)#A_translated_C_function

https://www.mkssoftware.com/docs/man1/bc.1.asp

http://www.numbertheory.org/gnubc/gnubc.html

https://www.gnu.org/software/bc/manual/html_mono/bc.html