bc

langage de calcul arbitraire

bc est un langage qui supporte les nombres à précision arbitraire avec exécution intéractive des déclarations. Il y'a certaines similarités avec le langage C. **bc** commence par traiter le code depuis tous les fichiers de la ligne de commande, puis il lit l'entrée standard.

OPTIONS

-i, –interactive Mode interactif

-l, –mathlib Définis la librairie standard

-w, –warn Donne des alertes pour les extensions à bc POSIX

-q, –quiet N'affiche pas le message de bienvenu

Nombres

C'est l'élément de base de bc. Les nombres sont de précision arbitraire. Tous les nombres sont représentés en interne en décimal et tous les calculs sont fait en décimal. Il y'a 2 attributs de nombres. la longueur et l'échelle. La longueur est le nombre total de chiffres et l'échelle et le nombre total après le point décimal. Par ex :

.000001 a une longueur de 6 et une échelle de 6.

1935.000 a une longueur de 7 et une échelle de 3.

Variables

Les nombres sont stockés dans 2 types de variables, les variables simples et le tableaux. Ces deux types sont nommés. Les noms commencent avec une lettre suivi par des lettres, chiffres et '_'. Toutes les lettres doivent être en minuscules.

Il y'a 4 types spéciaux de variables, **scale**, **ibase**, **obase** et **base**. **scale** définis la manière dont certaines opérations utilisent les chiffres après le point décimal (défaut : 0). **ibase** et **obase** définissent le base de conversion pour les nombres entrant et sortant (défaut : 10). **last** est une variable qui a la valeur du dernier nombre affiché.

Commentaires

Les commentaires dans bc commencent avec **/*** et se terminent par ***/**.

peut également être utilisé pour les commentaire sur une seule ligne.

Expressions

Les nombres sont manipulés par des expressions et des déclarations. Il n'y a pas de programme main. À la place, le code est exécutés tel qu'il est rencontré.

Une simple expression est simplement une constante. bc convertis les constantes en nombres décimal en utilisant la base d'entrée courante, spécifiée par la variable ibase. Les valeurs légales de ibase vont de 2 à 16. Les nombres en entrée peuvent contenir les caractères 0-9 en A-F. Pour les nombres à plusieurs chiffres, bc change tous les chiffres supérieurs ou égal à la valeur de ibase-1. Cela fait du nombre FFF toujours plus large que 3 chiffre de la base d'entrée

Les expressions sont similaires à d'autre langages de haut niveau. Vu qu'il n'y a qu'un type de nombre, il n'y a pas de règles pour mixer les types. À la place, il y'a des règles sur l'échelle des expressions. Toute expression a une échelle. L'échelle est dérivée des nombres originaux, l'opération est effectuée dans tous les cas, la valeur de la variable scale. les valeurs légales de scale vont de 0 au nombre maximum représentable par un entier C.

Dans les descriptions suivantes d'expressions légales, expr réfère à une expression complète et var réfère à une variable simple ou un tableau.

Une variable simple est juste : name
et un tableau est spécifié : name[expr]

- expr Le résultat est la négation de l'expression
++ var La variable est incrémenté de un et la nouvelle valeur est le résultat de l'expression
– var La variable est décrémenté de un et la nouvelle valeur est le résultat de l'expression
var ++ Le résultat de l'expression est la valeur de la variable puis la variable est incrémenté
var – Le résultat de l'expression est la valeur de la variable puis la variable est décrémenté
expr + expr Le résultat de l'expression est la somme des 2 expressions
expr - expr Le résultat de l'expression est la différence des 2 expressions
expr * expr Le résultat de l'expression est le produit des 2 expressions
expr / expr Le résultat de l'expression est de quotient des 2 expressions. l'échelle du résultat est la valeur de la variable scale
expr % expr Le résultat de l'expression est le reste
expr ^expr Le résultat de l'expression est la valeur de la première élevé à la seconde. La seconde expression doit être un entier.
(expr) Altère la précédence standard pour forcer l'évaluation de l'expression
var = expr Assigne la valeur de l'expression à la variable
var <op>=expr Est équivalent à "var = var <op> expr" à l'exception que var est évalué une seule fois. Peut faire une différence si var est un tableau

Les expressions relationnels sont un type d'expression spécial qui s'évalue toujours à 1 ou 0. Les opérateurs relationnels sont :

expr1 < expr2 Le résultat est 1 si expr1 est strictement inférieur à expr2
expr1 <= expr2 Le résultat est 1 si expr1 est inférieur ou égal à expr2
expr1 > expr2 Le résultat est 1 si expr1 est strictement supérieur à expr2
expr1 >= expr2 Le résultat est 1 si expr1 est supérieur ou égal à expr2
expr1 == expr2 Le résultat est 1 si expr1 est égal à expr2
expr1 != expr2 Le résultat est 1 si expr1 est différent de expr2

Les opérations booléennes sont aussi légales. Les opérations booléennes sont :

!expr Le résultat est 1 si expr est 0
expr && expr Le résultat est 1 si les 2 expressions ne valent pas 0
expr || expr Le résultat est 1 si au moins une des expressions ne vaut pas 0

La précédence des expressions est la suivante :

||

&&

!

Opérateurs relationnels

Opérateurs d'assignement

+ et -

^

- unaire

– et ++

Cette précédence à été choisie pour être conforme avec la version POSIX de bc. Cela occasionne quelques différences avec d'autres langages. par exemple, **a = 3 < 5** va assigner 3 à a puis comparer 3 à 5.

Fonctions spéciales

length (expression) retourne le nombre de chiffres dans l'expression

read ( ) Lit un nombre sur l'entrée standard

scale (expression) retourne le nombre de chiffres aprés le point décimal dans l'expression

sqrt (expression) retourne la racine carré de l'expression. Si l'expression est négative, retourne une erreur

Déclarations

Les déclarations fournissent le séquençage de l'évaluation d'expression. Dans bc les déclarations sont exécutés dès que possible. ' ;' et newline sont utilisés comme séparateur de déclaration. À cause de l'exécution immédiate, les newline sont très important. Il est possible de cacher un newline en utilisant un '\'. La séquence "\<nl>", où <nl> est le newline apparaît à bc comme un espace blanc au lieu d'un newline.

expression Si l'expression commence avec "<variable> <assignement> ...", c'est considéré comme une déclaration d'assignement. Si l'expression n'est pas un assignement, elle est évaluée et affichée sur la sortie, suivi d'un newline.

string La chaîne est affichée sur la sortie. Les chaînes sont entre '"'

print list La déclaration print fournis une autre méthode de sortie. list est une liste de chaînes et d'expressions séparé par ','. Les expressions sont évaluées et leur valeur sont affichés et assigné à la variable last.

{ statement_list } Permet de regrouper les déclarations ensemble pour exécution

if (expression) statement1 [else statement2] Évalue expression et exécute statement1 ou statement2 en fonction

while (expression) statement Exécute la déclaration tant que expression est différent de 0

for ([expression1] ; [expression2] , [expression3]) statement expression1 est évalué avant la boucle, expression2 est évalué avant chaque exécution de la déclaration, expression3 est évalué avant chaque ré-évaluation de expression2.

le code suivant est équivalent :

```
expression1;
while (expression2) {
 statement,
 expression3;
}
```

break Force à sortir d'une déclaration while ou for

continue Force une nouvelle itération d'une déclaration for

halt force à quitter bc

return Retourne la valeur 0 depuis une fonction

return (expression) Retourne la valeur de l'expression depuis une fonction

Pseudo déclarations

Ces déclarations ne sont pas exécutées, leur fonctions sont effectées à la compilation

limits Affiche les limites locales définis par la version de bc.

quit Identique à halt' excepté qu'il est toujours effectué (ex : **if (0 = 1) quit** termine bc, ce qui n'est pas le cas de halt)

warranty Affiche une note de garantie

Fonctions

Les fonctions dans bc calculent toujours une valeur et la retourne à sont appelant. Les définitions des fonctions sont dynamiques dans le sens qu'une fonction est indéfinie jusqu'à ce qu'une définition soit rencontrée. Une nouvelle définition remplace la précédente.

Une fontion est définie comme suit :

```
define name ( parameters ) { newline
 auto_list statement_list
```

Un appel de fonction est une simple déclaration sous la forme :

"name(parameters)"

Les paramètres sont des nombres ou des tableaux. Dans la définition de la fonction, on peut définir 0 ou plusieurs paramètres séparé par des virgules. Le nombre de paramètres doit correspondre lors de l'appel de la fonction.

auto_list est une liste optionnelle de variables locales. La syntaxe est :**"auto name, ... ;"**. Chaque name est le nom d'une auto variable. Ces variables sont initialisées à 0 et utilisées durant l'exécution de la fonction. Les auto variables sont différentes des variables locales traditionnelles parce que si une fonction A appel une fonction B, B peut accéder aux auto variables de A. Vue que bc push ces variables dans la pile, bc supporte les fonctions récursives

Le corp de la fonction est une liste de déclaration bc. return termine la fonction et retourne la valeur 0 à la fonction appelante, ou la valeur de l'expression, en fonction de la variante de return utilisée.

Les fonctions changene également l'utilisation de la variable **ibase**. Toutes les constantes dans le corp de la fonction seront convertis avec **ibase** au moment de l'appel de la fonction. Les changements de **ibase** seront ignorés durant l'exécution de la fonction excepté pour la fonction standard **read**.

De nombreuses extensions ont été ajoutés aux fonctions. Le format de la définition a été légèrement relaxé. Cette version de bc permet plusieurs newlines avant et après le'{' ouvrante de la fonction.

Par exemple, les définitions suivantes sont légales

```
define d (n) { return (2*n); }
define d (n)
{ return (2*n); }
```

Les fonctions peuvent être définies en **void**. Cette fonction ne retourne pas de valeur. Le mot clé **void** est placé entre **define** et le nom de la fonction.

Par exemple, la session suivante

```
define py (y) { print "-->", y, "<--", "0; }
define void px (x) { print "-->", x, "<--", "0; }
py(1)
-->1<--
0
px(1)
-->1<--
```

Librairie mathématique

Si bc est invoqué avec l'option **-l**, une librairie mathématique est préchargée et l'échelle par défaut est 20. La librairie mathématique définis les fonctions suivantes :

s (x) Le sinus de x, x est en radians

c (x) Le cosinus de x, x est en radians

a (x) Le arc tangente de x, retourne un radians

l (x) Le logarithme naturel de x

e (x) La fonction exponentiel du reste e à la valeur x

j (n,x) La fonction Bessel de l'ordre entier n de x

Exemples

Dans /bin/sh, le code qui suit va assigner la valeur de "pi" à la variable shell pi

pi=$(echo "scale=10 ; 4*a(1)" | bc -l)

La définition de la fonction exponentiel utilisée dans la librairie mathématique. et POSIX bc

```
scale = 20
 /*Uses the fact that e^x = (e^(x/2))^2
When x is small enough, we use the series:
e^x = 1 + x + x^2/2! + x^3/3! + ...
/

define e(x) {
 auto a, d, e, f, i, m, v, z

 /*Check the sign of x.*/
 if (x<0) {
  m = 1
  x = -x
 }

 /*Precondition x.*/
 z = scale;
 scale = 4 + z + .44*x;
 while (x > 1) {
  f += 1;
  x /= 2;
 }
```

```
/*Initialize the variables.*/
 v = 1+x
 a = x
 d = 1

 for (i=2; 1; i++) {
  e = (a *= x) / (d *= i)
  if (e == 0) {
   if (f>0) while (f-) v = v*v;
   scale = z
   if (m) return (1/v);
   return (v/1);
  }
  v += e
 }
}
```

ce code utilise les extensions bc pour implémenter un simple programme de calcul de solde de chéquiers

```
scale=2
print "\nCheck book program!\n"
print " Remember, deposits are negative transactions.\n"
print " Exit by a 0 transaction.\n\n"
print "Initial balance? "; bal = read()
bal /= 1
print "\n"
while (1) {
 "current balance = "; bal
 "transaction? "; trans = read()
 if (trans == 0) break;
 bal -= trans
 bal /= 1
}
quit
```

Définition d'une fonction factorielle récursive

```
define f (x) {
 if (x <= 1) return (1);
 return (f(x-1) * x);
}
```

Options Readline et Libedit

GNU bc peut être compilé pour utiliser la librairie d'entrée GNU readline ou BSD libedit. Cela permet à l'utilisateur d'éditer les lignes avant de les envoyer à bc. Cela permet également de conserver un historique. Quand cette option est sélectionné, bc a une variable spéciale supplémentaire, **history**

Limites

Les limites suivantes sont dans le pré-processeur bc. max input base : 16

BC_BASE_MAX base de sortie max, défaut : 999.

BC_DIM_MAX Limite arbitraire de 65535

BC_SCALE_MAX Le nombre de chiffres avant et après la virgule sont chacun limité par INT_MAX

BC_STRING_MAX La limite du nombre de caractères dans une chaîne est de INT_MAX caractères

exponent La valeur de l'exposant (^) est limité à LONG_MAX

variable names La limite du nombre de noms unique est de 32767 pour chaque variable simples, tableaux et fonctions

Variables d'environnement

POSIXLY_CORRECT Identique à -s, conforme bc à POSIX

BC_ENV_ARGS Autre mécanisme pour donner les arguments à bc, ces arguments sont traités en premier

BC_LINE_LENGTH Spécifie le nombres de caractères sur une ligne de sortie. À 0, désactive la sortie multi-ligne. Une valeur inférieur à 3 définis la longueur de ligne à 70