III.2.2. Reprezentarea în virgulă mobilă

Una dintre cele mai răspândite reprezentări internă (în PC-uri) a numerelor reale este ***reprezentarea în virgulă mobilă***. Reprezentarea în virgulă mobilă presupune existența unei ***baze b*** (întotdeauna presupusă pară) şi a unei ***precizii p***. Un număr în virgulă mobilă este un număr de forma

$$\pm(\alpha_0 + \frac{\alpha_1}{b} + \frac{\alpha_2}{b^2} + \ldots + \frac{\alpha_{p-1}}{b^{p-1}})b^E,\ \alpha_k \in \{0, 1, \ldots b-1\} \text{ pentru orice } k = \overline{0, p-1},\ E \in \mathbf{Z}.$$

Mai precis, denumirea de număr în virgulă mobilă va fi utilizată pentru numerele reale care se reprezintă exact sub forma de mai sus. În această reprezentare α_0, α_1, …, α_{p-1} se numesc ***cifre semnificative***. Fiecărei reprezentări în virgulă mobilă i se asociază două numere întregi, E_{min} şi E_{max}, ce reprezintă ***valorile limită*** permise pentru exponentul E ($E_{min} \le E \le E_{max}$). Tabelul de mai jos exemplifică cei patru parametri (baza, precizia, valorile limită ale exponentului) ce caracterizează reprezentarea în virgulă mobilă în diverse sisteme

Sistem	baza b	precizia p	E_{min}	E_{max}
IEEE single-precission	2	24	-126	127
IEEE double-precission	2	53	-1022	1023
Cray	2	48	-16383	16384
calculator HP	10	12	-499	499
mainframe IBM	16	6	-64	63

Reprezentarea în virgulă mobilă se numeşte ***normalizată*** dacă se impune condiția ca cifra cea mai semnificativă α_0 să fie nenulă. Reprezentarea normalizată are următoarele avantaje:

- reprezentarea fiecărui număr este unică
- nu de pierd cifre pentru reprezentarea primele zerourilor de la dreapta virgulei

- în sistemele binare (corespunzătoare bazei b =2) prima cifră poate să nu mai fie stocată (deoarece este întotdeauna 1).

Restricția $\alpha_0 \neq 0$, face imposibilă reprezentarea lui zero. O reprezentare naturală a lui zero este $1{,}0 \cdot b^{E_{min}-1}$.

Numărul de numere în virgulă mobilă normalizată este

$$2(b-1)b^{p-1}(E_{max} - E_{min} +1).$$

Cel mai mic număr pozitiv normalizat se notează UFL (***underflow level***) şi este

$$UFL = b^{E_{min}}.$$

Cel mai mare număr normalizat se notează OFL (***overflow level***) şi este

$$OFL = (b-1 + \frac{b-1}{b} + \frac{b-1}{b^2} + \ldots + \frac{b-1}{b^{p-1}})\ b^{E_{max}}$$

$$= b^{E_{max}+1}(1 - \frac{1}{b^p}).$$

Ca urmare nu toate numerele reale sunt reprezentabile exact. Numerele prea mari pentru a fi reprezentate corespund unei depăşiri superioare de capacitate (***overflow***), iar numerele prea mici unei depăşiri inferioare de capacitate (***underflow***). Pentru a fi reprezentat un număr real ***x*** este aproximat cu un număr în virgulă mobilă pe care convenim să-l notăm ***fl(x)***. Aproximarea lui x prin fl(x) poartă numele de ***rotunjire***, iar eroarea introdusă de ***eroare de rotunjire***. Există mai multe modalități pentru rotunjire:

- ***trunchiere (rotunjire prin tăiere)***: se reţin primele p cifre din reprezentarea normalizată a lui $x = \pm(\alpha_0 + \frac{\alpha_1}{b} + \frac{\alpha_2}{b^2} + \ldots + \frac{\alpha_{p-1}}{b^{p-1}} + \ldots)b^E$; deci

$$fl(x) = \pm(\alpha_0 + \frac{\alpha_1}{b} + \frac{\alpha_2}{b^2} + \ldots + \frac{\alpha_{p-1}}{b^{p-1}})b^E.$$

- ***rotunjire la cel mai apropiat număr în virgulă mobilă (rotunjire la par)***: fl(x) este cel mai apropiat număr în virgulă mobilă de x; în caz de egalitate (dacă există două numere în virgulă mobilă egal depărtate de x) se consideră acel număr în virgulă mobilă a cărui ultimă cifră este pară.

Rotunjirea la par determină o acuratețe mai mare a reprezentării. Acuratețea sistemului în virgulă mobilă este caracterizată de aşa numita ***precizie a maşinii*** (sau

epsilon maşină), notată ε_{mach}. Precizia a maşinii este definită ca cel mai mic număr pozitiv ε cu proprietatea că

$$fl(1.+ \varepsilon) > 1.$$

Dacă regula de rotunjire este trunchierea atunci

$$\varepsilon_{mach} = b^{1-p},$$

iar dacă regula de rotunjire este rotunjirea la par atunci

$$\varepsilon_{mach} = \frac{1}{2} b^{1-p}.$$

Eroarea relativă maximă cu care fl(x) aproximează x este dată de

$$\left| \frac{fl(x) - x}{x} \right| \leq \varepsilon_{mach}.$$

Deşi amândouă sunt "mici", precizia maşinii (ε_{mach}) şi cel mai mic număr pozitiv normalizat UFL (în reprezentare în virgulă mobilă fixată) nu trebuie confundate. De obicei $E_{min} < -p$ şi deci între ele există relaţia

$$0 < UFL < \varepsilon_{mach} < OFL.$$

Fie x un număr real aproximat de

$$fl(x) = \pm(\alpha_0 + \frac{\alpha_1}{b} + \frac{\alpha_2}{b^2} + \ldots + \frac{\alpha_{p-1}}{b^{p-1}})b^E.$$

Exponentul E poate lua atât valori pozitive cât şi valori negative. Cel mai adesea exponentul este “decalat “ şi reprezentat ca un număr întreg pozitiv (fără semn). Aceasta deoarece ordinea lexicografică (stabilită între şirurile de cifre din reprezentare) şi ordinea naturală sunt compatibile în cazul numerelor întregi fără semn. În consecinţă, compararea exponenţilor (şi a numerelor reale corespunzătoare) poate fi făcută eficient. Astfel reprezentarea internă a unui număr real x aproximat prin $fl(x) = \pm(\alpha_0 + \frac{\alpha_1}{b} + \frac{\alpha_2}{b^2} + \ldots + \frac{\alpha_{p-1}}{b^{p-1}})b^E$ se face sub forma

s	ed	$\alpha_0\alpha_1\ldots\alpha_{p-1}$

unde s este semnul lui x (se completează cu 0 dacă semnul este + şi cu 1 dacă semnul este -) iar ed este exponentul obținut prin adunarea unui decalaj D la exponentul E:

$$ed = E + D.$$

III.2.3. Standardul IEEE-754

IEEE este acronim pentru **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers, o organizație ce are drept principal scop elaborarea standardelor pentru produsele hardware şi software. Standardul IEEE-754 se referă la aritmetica în virgulă mobilă în sistemele binare. Acest standard precizează formatul de reprezentare în memorie în simplă şi dublă precizie a unui număr real. Reprezentarea se face în virgulă mobilă normalizată:

$$x \approx fl(x) = \pm (1 + \frac{\alpha_1}{2} + \frac{\alpha_2}{2^2} + \ldots + \frac{\alpha_{p-1}}{2^{p-1}})2^E,\ p = 24, 53.$$

Sunt admise şi aşa numitele ***numere denormalizate*** ("***denormalized floating-point numbers***"):

$$\pm (0 + \frac{\alpha_1}{2} + \frac{\alpha_2}{2^2} + \ldots + \frac{\alpha_{p-1}}{2^{p-1}})2^E,\ p = 24, 53,$$

cu cel puțin una dintre cifrele binare $\alpha_1, \alpha_2, \ldots, \alpha_{p-1}$ nenule.

Standardul IEEE-754 defineşte două valori speciale pentru situații excepționale:

- ***Inf***, pe post de "infinit" ("infinity"), pentru rezultatul împărțirii unui număr finit la zero.
- ***NaN***, pe post de "non-număr" ("not a number"), pentru rezultatul următoarelor operații
  - ▷ Adunare : Inf + (-Inf)
  - ▷ Înmulțire: 0·Inf
  - ▷ Împărțire: 0/0 sau Inf/Inf
  - ▷ Calculul restul împărțirii unui număr x la 0 sau a lui Inf la x
  - ▷ Calculul rădăcinii pătrate $\sqrt{x}$ pentru $x < 0$.

Scopul acestor valori este acela de a permite continuarea calculului.

Un număr în virgulă mobilă $\pm(\alpha_0 + \frac{\alpha_1}{2}+\frac{\alpha_2}{2^2}+\ldots+\frac{\alpha_{p-1}}{2^{p-1}})2^E$ se reprezintă intern conform IEEE-754 sub forma

s	ed	$\alpha_1 \ldots \alpha_{p-1}$

unde pentru s se rezervă un bit ce se completează cu 0 dacă numărul este pozitiv şi cu 1 dacă numărul este negativ, iar pentru exponentul decalat ed se rezervă k biți (k=8, 11). Decalajul considerat este $D = 2^{k-1} - 1$, deci

$$ed = E + 2^{k-1} - 1,$$

Pe k biți se pot reprezenta ca numere întregi fără semn 2^k valori, de la 0 la $2^k - 1$. Valorile 0 şi $2^k - 1$ sunt rezervate pentru numerele denormalizate şi pentru valorile speciale Inf şi Nan. Deci pentru un număr în virgulă mobilă normalizată trebuie îndeplinită condiția $1 \le ed \le 2^k-2$. De aici rezultă că $-2^{k-1}+2 \le E \le 2^{k-1}-1$. De exemplu, pe k = 8 biți se pot reprezenta numere întregi fără semn de la 0 la 255. Decalajul considerat este $2^7 - 1 = 127$, deci exponentul E ia valori de la – 126 la 127. Numărul de biți rezervați pentru exponent determină intervalul de numere reale reprezentabile în calculator. Numărul de biți rezervați pentru mantisă determină precizia de reprezentare (gradul de detaliere) a numerelor. Reprezentarea $\pm(\alpha_0 + \frac{\alpha_1}{2}+\frac{\alpha_2}{2^2}+\ldots+\frac{\alpha_{p-1}}{2^{p-1}})2^E$ fiind normalizată, există siguranța că $\alpha_0 = 1$, ceea ce permite omiterea sa (***bit ascuns***) pentru creşterea preciziei de reprezentare, dar complică prelucrarea informației.

Formatele de reprezentare a numerelor în virgulă mobilă (conform standardului IEEE 754) sunt:

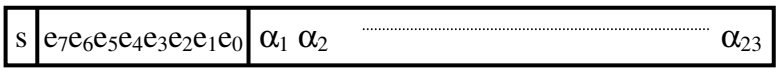
- ***simplă precizie*** (***single-precission***) pe 32 de biți:
  - 1 bit pentru semnul mantisei
  - 8 biți pentru exponentul decalat ($E_{min} = -126$, $E_{max} = 127$)
  - 23 biți pentru mantisă ($p = 24$, $\alpha_0 = 1$ se omite)
- ***dublă precizie*** (***double-precission***) pe 64 de biți
  - 1 bit pentru semnul mantisei
  - 11 biți pentru exponentul decalat ($E_{min} = -1022$, $E_{max} = 1023$)

- 52 biți pentru mantisă (p = 53, $\alpha_0 = 1$ se omite)

Regula de rotunjire este rotunjirea la par. Deci pentru

- simplă precizie, $\varepsilon_{mach} = 2^{-24} \approx 10^{-7}$ (7 cifre zecimale semnificative).
- dublă precizie, $\varepsilon_{mach} = 2^{-53} \approx 10^{-16}$ (16 cifre zecimale semnificative).

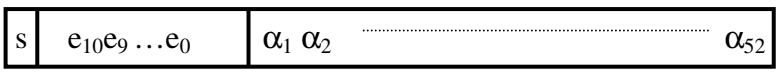
Considerăm o reprezentare în memorie, în simplă precizie:



Fie $ed = e_0 + e_1 2 + e_2 2^2 + \ldots + e_7 2^7$ şi $m = \frac{\alpha_1}{2} + \frac{\alpha_2}{2^2} + \ldots + \frac{\alpha_{23}}{2^{23}}$. Valoarea v reprezentată se determină după cum urmează:

- ▷ dacă $0 < ed < 255$, atunci $v = (-1)^s \cdot (1 + m) \cdot 2^{ed - 127}$.
- ▷ dacă $ed = 0$, $\alpha_k = 0$ pentru orice $k = \overline{1,23}$ şi $s = 0$, atunci $v = 0$.
- ▷ dacă $ed = 0$, $\alpha_k = 0$ pentru orice $k = \overline{1,23}$ şi $s = 1$, atunci $v = -0$.
- ▷ dacă $ed = 0$ şi există $\alpha_k \neq 0$, atunci $v = (-1)^s \cdot m \cdot 2^{-126}$; v este o valoare denormalizată
- ▷ dacă $ed = 255$, $\alpha_k = 0$ pentru orice $k = \overline{1,23}$ şi $s = 0$, atunci $v = Inf$.
- ▷ dacă $ed = 255$, $\alpha_k = 0$ pentru orice $k = \overline{1,23}$ şi $s = 1$, atunci $v = -Inf$.
- ▷ dacă $ed = 255$ şi există $\alpha_k \neq 0$, atunci $v = NaN$.

Fie reprezentarea în memorie, în dublă precizie:



Fie $ed = e_0 + e_1 2 + e_2 2^2 + \ldots + e_{10} 2^{10}$ şi $m = \frac{\alpha_1}{2} + \frac{\alpha_2}{2^2} + \ldots + \frac{\alpha_{52}}{2^{52}}$. Valoarea v reprezentată se determină după cum urmează:

- ▷ dacă $0 < ed < 2047$, atunci $v = (-1)^s \cdot (1 + m) \cdot 2^{ed - 1023}$.
- ▷ dacă $ed = 0$, $\alpha_k = 0$ pentru orice $k = \overline{1,52}$ şi $s = 0$, atunci $v = 0$.

- ▷ dacă ed = 0, $\alpha_k = 0$ pentru orice k=$\overline{1,52}$ şi s = 1, atunci v = - 0.
- ▷ dacă ed = 0 şi există $\alpha_k \neq 0$, atunci $v = (-1)^s \cdot m \cdot 2^{-1022}$; v este o valoare denormalizată
- ▷ dacă ed = 2047, $\alpha_k = 0$ pentru orice k=$\overline{1,52}$ şi s = 0, atunci v = Inf.
- ▷ dacă ed = 2047, $\alpha_k = 0$ pentru orice k=$\overline{1,52}$ şi s = 1, atunci v = -Inf.
- ▷ dacă ed = 2047 şi există $\alpha_k \neq 0$, atunci v = NaN.

***Exemple**:*

Să se reprezinte în simplă precizie numerele: 228,15 - 27, 25 0,1 1,2

x = 228,15

x = 228 + 0,15

$228 = 128 + 64 + 32 + 4 = 2^7 + 2^6 + 2^5 + 2^2 = 11100100_2$

$0,15 \cdot 2 = 0,30 = 0 + 0,3$

$0,3 \cdot 2 = 0,6 = 0 + 0,6$

$0,6 \cdot 2 = 1,2 = 1 + 0,2$

$0,2 \cdot 2 = 0,4 = 0 + 0,4$

$0,4 \cdot 2 = 0,8 = 0 + 0,8$

$0,8 \cdot 2 = 1,6 = 1 + 0,6$

⋮

x = 11100100,00100110011001…

Forma normalizată:

$$x = 0,11100100001001100 1\ldots \cdot 2^8 = 1,1100100001001100 1\ldots \cdot 2^7$$

$ed = 7 + 2^{8-1} - 1 = 135$, $ed_2 = 10000110_2$

m = 11001000010011001100110 [011]

(am omis primul bit =1, iar cei trei biți din paranteză sunt utilizați pentru rotunjire la par)

$fl(x) = 1, 11001000010011001100110 \cdot 2^8$

Reprezentare în virgulă mobilă, simplă precizie, (cu bit ascuns) pentru 228,15:

0	100	0011	0	110	0100	0010	0110	0110	0110
	4	3		6	4	2	6	6	6

Deci reprezentării cu bit ascuns a lui 228,15 îi corespunde 43642666 în hexazecimal.

$x = -27,25$

$|x| = 27 + 0,25$

$27 = 16 + 8 + 2 + 1 = 2^4 + 2^3 + 2^1 + 2^0 = 11011_2$

$0,25 = 2^{-2} = 0,01_2$

$x = 11011,01$

Forma normalizată: $x = 0,1101101 \times 2^5 = 1,101101 \times 2^4$

$ed = 4 + 2^{8-1} - 1 = 131$, $ed_2 = 10000011_2$

Reprezentare în virgulă mobilă, simplă precizie (cu bit ascuns) pentru –27,25:

1	1 0 0 0 0 0 1 1	1 0 1 1 0 1 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
C	1	D	A	0	0	0	0

Deci reprezentării cu bit ascuns a lui -27,25 îi corespunde C1DA0000 în hexazecimal.

$x = 0,1$

$0,1 \cdot 2 = 0,2$

$0,2 \cdot 2 = 0,4$

$0,4 \cdot 2 = 0,8$

$0,8 \cdot 2 = 1,6 = 1 + 0,6$

$0,6 \cdot 2 = 1,2 = 1 + 0,2$

$0,2 \cdot 2 = 0,4$

$0,1_{10} = 0, 00011001100110011\ldots$

$x = 0, 110011001100\ldots \cdot 2^{-3} = 1,10011001100110011001100\ 110\ldots \cdot 2^{-4}$

$fl(x) = 1, 10011001100110011001101 \cdot 2^{-4}$

(după cei 23 de biți ai mantisei urmează 110, şi deci rotunjirea se face prin adăugarea unei unități).

$ed = -4 + 2^{8-1} - 1 = 123 = 2^6 + 2^5 + 2^4 + 2^3 + 2 + 1$, $ed_2 = 1111011_2$

Reprezentare în virgulă mobilă, simplă precizie (cu bit ascuns) pentru 0,1:

0	0 1 1 1 1 0 1 1	1 0 0 1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 1
3	D	C	C	C	C	C	D

Deci reprezentării cu bit ascuns a lui 0,1 îi corespunde 3DCCCCCD în hexazecimal.

x = 1,2

$1,2 = 1 + 0,2$

$0,2 \cdot 2 = 0,4$

$0,4 \cdot 2 = 0,8$

$0,8 \cdot 2 = 1,6 = 1 + 0,6$

$0,6 \cdot 2 = 1,2 = 1 + 0,2$

$0,2 \cdot 2 = 0,4$

$x = 1, 0011001100110011\ldots$

$x = 1,00110011001100110011001\ 100\ldots \cdot 2^0$

$fl(x) = 1,00110011001100110011010 \cdot 2^0$

(după cei 23 de biți ai mantisei urmează 100, deci rotunjirea se face astfel încât ultimul bit să aibă valoare pară).

$ed = 0 + 2^{8-1} - 1 = 127 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2 + 1$, $ed_2 = 1111111_2$

Reprezentare în virgulă mobilă, simplă precizie (cu bit ascuns) pentru 1,2:

0	0 1 1	11 1 1	1	0 0 1	1 0 01	1 0 01	1 0 01	1 0 01	1 0 1 0
	3	F		9	9	9	9	9	A

Deci reprezentării cu bit ascuns a lui 1,2 îi corespunde 3F99999A în hexazecimal.

Următorul program în C verifică reprezentările de mai sus.

```
#include <stdio.h>
#include <conio.h>
void main(){
   long int *i;
   float f1=228.15,f2=-27.25, f3=0.1, f4=1.2;
   clrscr();
   i=(long int*) &f1;
   printf("\nNumar in virgula mobila:%f\n\tFormat intern %08lX
(hexazecimal)",f1,*i);
   i=(long int*) &f2;
   printf("\nNumar in virgula mobila:%f\n\tFormat intern %08lX
(hexazecimal)",f2,*i);
```

```
      i=(long int*) &f3;
   printf("\nNumar in virgula mobila:%f\n\tFormat intern %08lX
(hexazecimal)",f3,*i);
      i=(long int*) &f4;
   printf("\nNumar in virgula mobila:%f\n\tFormat intern %08lX
(hexazecimal)",f4,*i);
     getch();
}
```

Programul afişează

```
Numar in virgula mobila: 228.149994
      Format intern 43642666 (hexazecimal)
Numar in virgula mobila: -27.250000
      Format intern C1DA0000 (hexazecimal)
Numar in virgula mobila: 0.100000
      Format intern 3DCCCCCD (hexazecimal)
Numar in virgula mobila: 1.200000
      Format intern 3F99999A (hexazecimal)
```

III.2.4. Aritmetica în virgulă mobilă

Din secțiunea precedentă rezultă că nu toate numerele reale pot fi reprezentate exact într-un sistem în virgulă mobilă. De asemenea în urma evaluării unei expresii ai cărei operanzi sunt reprezentabili rezultatul obținut nu este neapărat reprezentabil. În mod ideal

$$x \text{ flop } y = fl(x \text{ op } y)$$

unde op este un operator binar (+, - , *, /), iar flop desemnează corespondentul operatorului respectiv în aritmetica în virgulă mobilă. Sistemele ce satisfac standardul IEEE-754 ating acest ideal în situația în care x op y se găseşte în intervalul de numere reale reprezentabile [UFL, OFL].

Depăşirea superioară de capacitate (overflow) cauzează de obicei probleme mai serioase decât depăşirea inferioară de capacitate (underflow), deoarece nu există nici o aproximație bună pentru un număr real oarecare "mare". Un număr

real foarte mic poate fi în mod rezonabil aproximat cu zero. Pe multe sisteme de calcul depăşirea superioară de capacitate este fatală, în timp ce în caz de depăşire inferioară de capacitate, numărul respectiv este asociat cu zero.

Anumite legi ale aritmeticii reale nu sunt valabile într-un sistem în virgulă mobilă. Astfel adunarea şi înmulţirea în virgulă mobilă sunt comutative, dar nu asociative. De exemplu, dacă ε este un număr pozitiv mai mic decât ε_{mach}, dar mai mare decât $\varepsilon_{mach}/2$, atunci

$$(1 + \varepsilon) + \varepsilon = 1, \text{ iar } 1 + (\varepsilon + \varepsilon) > 1.$$

Rezultatul unei operaţii în virgulă mobilă poate să fie semnificativ diferit faţă de rezultatul aceleaşi operaţii în aritmetica exactă. Să considerăm numărul real $x = \frac{1}{10}$. Se reprezintă în baza 2, prin x = 0,0001100110011…=1, $1001100110011001100110101\ldots 2^{-4}$. În simplă precizie este aproximat de fl(x) = 1, $1001100110011001100110101 \cdot 2^{-4}$, ceea ce introduce o eroarea de 0.00000000000000000000000011001100 în binar sau aproximativ 0.000000047 în zecimal. Programul în C de mai jos pune în evidenţă cum se propagă această eroare prin înmulţire:

```
#include <stdio.h>
#include <conio.h>
void main(){
      float f=1./10,z=10;
      int i;
      clrscr();
      for(i=1;i<33;i*=2){
      printf("\n0.1*10^%d-10^%d = %f ",i+1,i,f*(z*10)-z);
      z=z*z;
      }
      getch();
}
```

Programul afişează

```
0.1*10^2 - 10^1 = 0.000000
0.1*10^3 - 10^2 = 0.000001
0.1*10^5 - 10^4 = 0.000149
```

```
0.1*10^9 - 10^8 = 1.490116
0.1*10^17 - 10^16 = 149011616.000000
0.1*10^33 - 10^32 = 14901161688288353600000000.000000
```

Membrul drept ar fi trebuit să fie 0, deoarece $\frac{1}{10}\cdot 10^{i+1} - 10^{i} = 0$.

În cazul scăderii a două numere reale x şi y, poate apărea următorul fenomen (***catastrophic cancellation***)

$$\left|\frac{(\mathrm{fl}(x)-\mathrm{fl}(y))-\mathrm{fl}(x-y)}{\mathrm{fl}(x-y)}\right| \succ \varepsilon_{mach},$$

dacă fl(x) este egal (sau foarte apropiat de) fl(y). În următorul program (în C) aproximăm sin(x) printr-o sumă parțială a seriei

$$\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+1}$$

Seria fiind alternantă şi convergentă, o sumă parțială de ordin n, aproximează suma seriei (i.e. sin(x)) cu o eroare absolută maximă de $\frac{|x|^{2n+1}}{(2n+1)!}$. Programul de mai jos reprezintă versiunea în C pentru calculul sumei care aproximează sin(x):

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(){
      float x,s,t,eps,x2;
      int i,n;
       clrscr();
       printf("x=");scanf("%f",&x);
       printf("Eroarea=");scanf("%f",&eps);
       t=x;s=0;i=1; x2=x*x;
       while (fabs(t)>=eps){
       s+=t;printf("\n%f",s);
       t=-t*(x2/(4*i*i+2*i));
       i++;
       }
       printf("\nsin(%f) = %f",x,s);
       printf("\nsin(%f) = %f" ,x,sin(x));
```

```
        getch();
}
```

Pentru x=2 şi eroare 10^{-7} se obține aproximația 0.909297 corectă a lui sin(2). Pentru x = 40 şi eroare 10^{-7} se obține aproximația 523443136.0 a lui sin(40) ! Valoarea corectă este 0.745113…Acest rezultat se datorează fenomenului de reducere (catastrophic cancellation).