ALGORITHM FOR LONG DIVISION AND ITS IMPLEMENTATION IN C LANGUAGE

Nicolai FALICO, sup. lec., PhD; Mihail KULEV, assoc. prof., PhD

Technical Univercity of Moldova

Abstract: *In the paper an effective algorithm for long division and its implementation in C language, developed by authors, has been presented. The proof of the correctness of algorithm is shown. The long number is implemented as the array of integers: a[0] is the number of digits , a[1] is the lowest digit and a[a[0]] is the highest, and highest digit is followed by zero. Program elaborated does not need normalization of dividend and divisor, as well as renormalization of residue.*

Keywords: *Algorithm for long division, program in C language, array of integers pointers.*

1. Introduction

Algorithms for long division are well known (see for example review in the fundamental work of Knut [1]). However, simple algorithms are not effective: they need much time for execution. The main difficulty in the effective algorithms is finding the correspondent digit of quotient. Some authors suggest finding this digit with the help of half division [2] that is simple for writing program, but the program becomes long and slow. Knut [1] very carefully proves simple formula that helps to find the good approximation for digits of quotient. This formula uses two digits of dividend and one of divisor. But in the following Knut [1] uses more complicated and more effective formula. His program written on MIX language contains more than hundred lines of code and not works when divisor is one digit. In that case Knut suggests using another program.

2. Algorithm description

In this paper another formula is proved:

```
                (double)o*o* *pa1 + o* *(pa1-1) + *(pa1-2)
      *prezc=  -----------------------------------------------------
                            o* *pa2 + *(pa2-1) + 1
```

It is as effective as in Knut [1] but needs no normalization of dividend and divisor and no renormalization of residue. The proof is given in the next paragraph of the paper and is not simple, but anybody can easily check the program using correspondent checking program and generating random inputs. We must mention, that the close algorithms were suggested by another authors (see, for example, [3, 4]). Program elaborated by authors is written in C (the base function has only 20 statements) and is based on special structure for long numbers [2]. The long number is the array of integers: a[0] is the number of digits, a[1] is the lowest digit and a[a[0]] is the highest, and highest digit is followed by zero. For example, the number 123456789 with the base o = 10000 is equal to a = {3, 6789, 2345, 1, 0}. In compare with the programs from [2], written in Pascal and using the method of half division, the developed code uses formula for finding digits. This make program very fast. Moving pointers that is common for C makes program short and simple and in contrary to program of Knut does not need normalization of dividend and divisor, as well as renormalization of residue.

3. Proof of the algorithm correctness

Let`s give the proof of the correctness of algorithm. The following notations will be used

$$\alpha\text{=Aa =A*}10^k\text{ +a, }\beta\text{=Bb =B*}10^k\text{ +b.}$$

Let's consider **$0<=a,b<10^k$, A<10*B, B>=10 and A/(B+1)=Q.q.**
It is not difficult to show that **Q*α<β.**

Let's proof that $(Q+2)^* \beta>\alpha$.

This is equivalent to $(A/(B+1)-0.q+2)^*)^* \beta>\alpha$, or that is the same

$$(A+(1-0.q)*(B+1)+B+1 \ )* \beta>\alpha*(B+1) \qquad (1)$$

Let's proof the following inequality. From that will be followed (1)

$$(A+B+1)* \ \beta>\alpha*(B+1) \qquad (2)$$

Inequality (2) is equivalent to $(A+B+1)*(B*10^k +b)> (A*10^k +a)*(B+1)$ or

$$B^2*10^k +B*b +A*b + B*b <A* 10^k +a*B+a \qquad (3)$$

Let`s show that

$$B^2*10^k \ - A* 10^k -a>0 \qquad (4)$$

That is followed from $a <10^k$ and $10^k *(B^2 -A)>= 10^k *(10*B-A) >= 10^k$, because $10*b-A$ is integer and $A< 10*B$. Besides that

$$Bb= B*10^k +b>a*B \qquad (5)$$

Inequality (3) is followed from (4) and (5).

4. Implementation in C language

```
#define o 10000L
```

The base is defined. For 16 – bits compiler maximum is 10000 because multiple of two 5 digit numbers is greater than MAXLONG. For 32-bits compiler o can be 100 000 000.

```
void print_l(int *x,char *y){} void a_al(char *x,int *y){}
```

This functions converts numbers from the long mode to usual notation and backwards. They can be found anywhere.

```
int les(int *mid,int *a2){
int i=*a2;
if(mid[i+1])
return 0;
while(i&&mid[i]==a2[i])
i--;
if(i==0)
return 0;
else
return (mid[i]<a2[i]);
}
```

This function gets pointers to two long numbers and return 1 if the first number (*mid) is less than the second (*a2) and 0 in another case. Comparison begins from the highest digits and works i=*a2 times, that is equal to number of digits of (*a2). Function returns 0 if the number of digits of (*mid) is greater than (*a2).

```
void  sub(int *a1,int q,int *a2){
        int i=1;
        while(i<=*a2){
                a1{i}-=(long)q * a2[i]%o;
                a1[i+1]-=(long)q*a2[i]/o;
                while(a1[i]<0){
                        a1[i]+=o;
                        a1[i+1]-=1;
                }
                i++;
        }
while(i<=*a1){
while(a1[i]<0){
        a1[i]+=o;
        a1[i+1]-=1;
}
I++;
}
}
```

This function gets pointers on two long numbers and digit. Number (*a2) is multiplied by q and is subtracted from (*a1) after working this program.

```
void d_l(int *a1,*a2,int *rez){
/*1*/ int *prez,*pa1=a1+*a1, *pa2=*a2+a2,*mid;
/*2*/  if((((*a1)-(*a2))>=0){
/*3*/                   mid = a1+*a1- *a2;
                        *rez=*a1- *a2+1;
            }else{
/*4*/                   *rez=1;rez[1]=0;
                        Return;
            }
/*5*/ prez=rez+*rez;
/*6/ while(prez>rez){
/*7*/ if(*a2>pa1-mid || (*a2==pa1-mid) && les(mid,a2) ==1){
/*8*/ prez-=o;
/*9*/ mid--;                        //snosim sled cifru
/*10*/      continue;
   }
/*11/ if(*a2==pa1-mid){
                        (*a1)++;
/*12*/                  pa1++;                  //razriad
            }
/*13*/      *prez=( (double) o*o*(*pa1) + o* (*(pa1-1))+
                        *(pa1-2))/(o*(*pa2) + (*(pa2-1)) +1);
/*14*/      sub(mid,*prez,a2);
/*15*/      if(les(mid,a2)!=1){
/*16*/                  (*prez)++;
/*17*/                  sub(mid,1,a2);
            }
/*18*/      while(*pa1 == 0 && pa1>1){
                        pa1--;
                        --*a1;                  //nahozdenie chisla cifr v a1
            }
/*19/ prez--;
            mid--;
}
/*20*/      if (rez[*rez]==0)
                        --*rez;
}
```

This is the base function. It gets three pointers on log numbers. After it works in rez the quotient from dividing a1 on a2 will be written and in a1 will be residual. In line 1 pointers prez, pa1, pa2 on the highest ranks of numbers rez, a1, a2 are defined. Pointer mid is defined so that mid +1 is the position of digit in a1 from which the subtraction will begin. In operators 2-5 the initial values are written to that pointers and also initial number of digits of the result is calculated. This value may be changed in the operator 20. In cycle 6 array rez is calculated from highest ranks. At first in 7-10 condition if, if the correspondent digit will be 0, is checked and than using shift mid--, next digit of dividend is processed. Operator 13 is the core of suggested algorithm. In it it three digits of highest ranks of dividend and two of divisor are used. In operator 11 the number of digits of dividend and divisor is checked and if needs the highest digit of a1 becomes 0 due to increasing number of digits on 1. In 12 the pointer to highest digit is changed. In 14 the subtracting from the position mid +1 in dividend is processed. In 15 the condition of continuing of dividing is checked and if yes than in 16 the result is increased on 1 and in 17 the subtracting is made. Because of that in a1 the number of digits is changed and in 17-18 the number of digits and pointer on the highest digit is changed. In 19 the pointer of the result is changed.

References

1. Д.Э. Кнут ., Искусство программирования . т.2. Третье издание . Москва, 2000.
2. С. Окулов., Программирование в алгоритмах.,изд. Бином.Лаборатория знаний, 2002.
3. George E. Collins, David R. Musser: Analysis of the Pope-Stein Division Algorithm. Information Processing Letters, 151-155, V.6, N.5, October 1977.
4. E.V. Krishnamurthy, Salil K. Nandi: On the normalization requirement of divisor in divide-and-correct methods. Communications of the ACM (CACM), 809-813. V. 10, N. 12, December 1967.