Informe de trabajo

Ejercicio de entrega obligatoria TP0

22 de marzo de 2023

Facultad de Ingeniería UNLP
E0301 Introducción al Diseño Lógico
Curso 2023

G1

s

• Caciani Toniolo, Melina melicaciani@gmail.com	02866/1
• Chanquía, Joaquín joaquin.chanquia@alu.ing.unlp.edu.ar	02887/7
• Larsen, Mateo Emmanuel larsenmateo.ml@gmail.com	02993/7
• Ollier, Gabriel gabyollier@hotmail.com	02958/4

REQUERIMIENTOS

Se solicita escribir un programa en lenguaje C que calcule la suma de los números que se encuentran almacenados en un archivo de texto a razón de un número por fila. Los números en el archivo pueden ser positivos o negativos y están representados con 5 dígitos enteros y 4 dígitos decimales separados entre sí por el punto decimal.

Restricciones para tener en cuenta: se solicita que todos los cálculos se realicen utilizando operaciones y variables enteras. Incluso la impresión de los resultados y el ingreso de datos. Primero se debe representar el número leído en punto fijo, realizar la suma en esta representación y luego imprimir el resultado en decimal.

CUESTIONARIO

a) ¿Cuál sería la representación en punto fijo que se debe utilizar para almacenar los valores leídos?

La representación que se puede usar como mínimo tiene 17 bits para representar a la parte entera y 14 bits para la parte decimal, además de un bit de signo. Esto surge al tomar la potencia de 2 inmediatamente superior al máximo número que es posible leer desde el archivo (99999) la cual es 2^17=131072. Por lo que con 17 bits es posible representar la parte entera, agregando un bit para representar el signo. Para la parte fraccionaria se debe buscar un número potencia negativa de 2 menor que la resolución buscada, en este caso 0,0001, y se encuentra que la primera potencia menor a ese número es 2^14=1/16384=0,000061. Por lo que se pueden usar 14 bits para representar a la parte decimal del número, dando un total de 17+14+1= 32 bits para representar el número pedido. En C tanto un entero como un unsigned guardan 4 bytes, es decir 32 bits.

b) ¿Cuáles serían los números más pequeño y más grande que se pueden representar? ¿y el más pequeño en valor absoluto?

El número más pequeño que se puede representar es:
-2^17 - 2^-1 - 2^-2 - 2^-3 - ... - 2^-14= -131072,99994

El número más grande que se puede representar es:
2^17-1 + 2^-1 + 2^-2 + 2^-3 + ... + 2^-14= 131071,99994

El número más pequeño en valor absoluto es:
2^-14~=0,000061

c) ¿Se puede utilizar la misma representación para realizar la suma? ¿Qué inconveniente encuentra?

No se puede usar la misma representación para la suma porque si el programa llegara a encontrarse con un archivo grande con muchos números la suma de todos ellos puede superar la representación usada para un solo número.

d) Proponga una representación adecuada para realizar la suma, sin perder dígitos significativos ni precisión.

Para realizar la suma, una representación más adecuada podría ser mantener los bits de decimales y el de signos igual y reemplazar el unsigned con el uso de un long unsigned, por lo que se contaría con 64 bits; suficiente para sumar el peor caso de un archivo con todos los números máximos (99999,9999) de hasta 5,6*10^9 números, ya que se contarán con 49 bits para la parte entera (64-1-14=49) y 2^49/99999,9999=5,6*10^9

ESTRATEGIA UTILIZADA

- **Representación**: al principio nosotros pensamos en utilizar un campo de bits, representando en un solo registro todo el número, limitando los bits, tanto de signo, de la parte entera y de la decimal utilizando este método, para así poder tener solo los bits necesarios para cada parte. Luego de un tiempo de debate llegamos a la conclusión que el manejo de campos de bits iba a complicar de más al programa, por lo que decidimos que la mejor manera de leer al número en dos partes al principio, una representando a la parte entera y otra a la parte decimal. Teniendo estas dos partes y habiendo leído el signo con anterioridad pudimos crear un método encargado de armar un número representativo del leído desde el archivo. Este primero sumaba la parte entera y la desplazaba como para que no coincidiera con la parte decimal y luego mediante el uso de una máscara iba activando los bits necesarios para representar al número en punto fijo.

- **Detección de errores:** al encontrarnos con que el código no funcionaba correctamente y desconociendo los motivos, recurrimos a algunos métodos para poder darnos cuenta dónde estaba el problema. Principalmente nos ayudamos con la impresión en pantalla de nuestras variables, esta nos dejaba controlar paso a paso cada operación para ver si estábamos trabajando con los valores que creíamos o estábamos tomando un valor equivocado. También nos sirvió crear la función verBits la cual la utilizamos para que nos imprima la representación del número en binario y así poder trabajar mejor con el tema de las máscaras y conversiones por los tipos de datos utilizados.

```
el numero es 0012.3400
suma antes= 1358.0238
parc      = 0.0012
parc<<14  = 12.0000
suma+parc = 1370.0238
  22446318 = 00000001 01010110 10000000 11101110
      dec = 0.3400
      3400 = 00000000 00000000 00001101 01001000
      9999 = 00000000 00000000 00100111 00001111
suma +dec = 1370.3638
  22449718 = 00000001 01010110 10001110 00110110
dec es 3638 suma fin  = 1370.3638
  22449718 = 00000001 01010110 10001110 00110110

el numero es 7777.7777
```

```
void verBits(long valor){

    unsigned size = (sizeof(valor)*8);
    long mascara = 1 << (size-1);
    printf("%10u = ", valor);
    for (unsigned i=1; i <= size ; i++){
        printf( valor & mascara ? "1" : "0");
        valor <<= 1;
        if (i % 8 == 0) printf(" ");
    }
    printf("\n");
}
```

EXPLICACIÓN

```
while (!feof(archivo)){
    c = fgetc(archivo);
    if (c != '\n'){
        if (c == '-'){
            signo = 1;
            fscanf(archivo,"%u.%u",&ent, &dec);
        }else{
            signo = 0;
            fscanf(archivo,"%u.%u",&ent, &dec);
            ent += ((c-0x30)*10000);
        }
        num = representarNum(signo,ent,dec);
        suma += num;;
    }
    fgetc(archivo);
}
```

Uno de los primeros problemas con el que nos encontramos al desarrollar este programa fue que teníamos que distinguir si el número a sumar era positivo o negativo y no podíamos simplemente almacenar el número en un dato de tipo int (que contiene el signo) ya que este no contempla los -0. Para esto tomamos el primer carácter con la función fgetc(), si era un " - " leíamos el número normalmente y lo restamos a la suma total, de lo contrario significaba que ya habíamos tomado el primer dígito del número. Esto sería un inconveniente porque, según lo entendido, debíamos recorrer el archivo una sola vez, por lo que desplazarse una posición hacia atrás en el archivo y leer todo el número no estaba permitido. Al no poder implementar la solución más sencilla tuvimos que, con ayuda de la tabla ASCII, convertir el carácter en su correspondiente número; esto lo hicimos restando el número 30 hexadecimal (ya que el 0 es 30H, el 1 es 31H y así sucesivamente) para obtener su valor, luego leímos el resto del número y le sumamos el valor obtenido multiplicado por 10000 para representar realmente el dígito que habíamos obtenido, el cual estaba en la posición de la decena de mil.

```
int representarNum(unsigned signo,unsigned ent,unsigned dec){
    int num = ent<<BITS_DEC;
    int mask=1<<(BITS_DEC-1);
    dec *= 1000;
    int comp=5000000;
    for (int i=0;i<14;i++){
        if (dec > comp){
            dec -= comp;
            num += mask;
        }
        mask = mask >> 1;
        comp /= 2;
    }
    if (!(num&1)&&(dec>0)){
        num +=1;
    }
    if (signo){
        num = ~num;
        num += 1;
    }
    return num;
}
```

Este método se encarga de guardar la representación en punto fijo del número que se pasa en los parámetros. Primero guardando la parte entera del número y luego representando la parte decimal, utilizando para esto dos variables que van cambiando en cada iteración: mask, la cual se compone de todos ceros y un 1, originalmente ubicado en el primer bit decimal; y la variable comp, encargada de mantener una representación del valor al cual hace referencia el bit marcado por mask, el mismo que se compara con la parte decimal del número para ver si ese bit debe estar activo. Al finalizar se le cambia el signo al numero en caso de ser necesario.

El método imprimirSuma se encarga de imprimir en pantalla el número representado en punto fijo, siguiendo un método similar a representarNum. Revisando el número con una mascara y si se encuentra con un bit activo se suma a la parte decimal el comparador adecuado.

```
void imprimirSuma(TIPO_SUMA num){
    int signo = 0;
    unsigned ent, dec;
    if (num&0x8000000000000000){
        signo = 1;
        num -= 1;
        num = ~num;
    }
    int mask=1<<(BITS_DEC-1);
    int comp=5000000;
    for (int i=0;i<14;i++){
        if (num&mask){
            dec += comp;
        }
        mask = mask >> 1;
        comp /= 2;
    }
    dec /= 1000;
    ent = num>>BITS_DEC;
    printf("%c%05u,%04u\n",signo ? '-' : ' ',ent,dec);
}
```