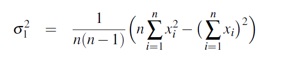
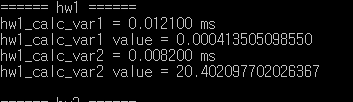
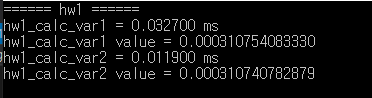
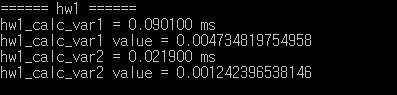
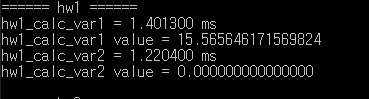
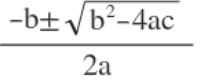
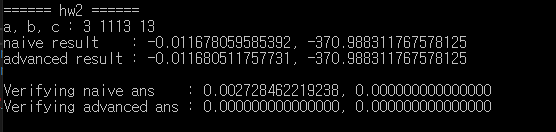
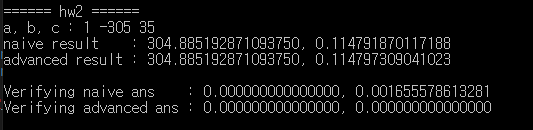
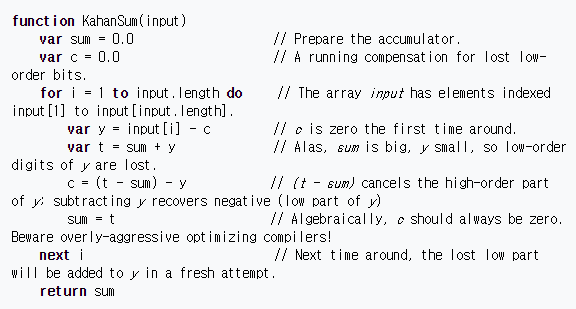
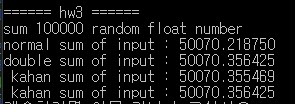
**고급 소프트웨어 실습1**

**10주차 : 코드 최적화 기법/부동소수점 연산**

20171640 박수진

1. 과제1  
   hw1\_calc\_var1은 식을 사용하고,  
   hw1\_calc\_var2는 식을 사용하여 계산하였다.  
     
     
   n = 1000일 때 float 자료형을 사용하면 위와 같이 두 분산의 값이 매우 다르게 나타난다. 컴퓨터 상에서 소수의 표현에 한계가 있기 때문에 비슷한 수의 뺄셈을 할 경우 잘못된 값이 나올 수 있다. Float는 32bit(4bytes)로 끊어서 표현하기 때문에 표현의 범위가 좁아 계산양이 많아지면 오차가 엄청 커질 수 있다. 그래서 오차를 줄이기 위해 float 대신 표현의 범위가 더 큰 double형(8bytes)을 사용하였다.  
     
   double형을 사용하여 n = 1000일 때의 계산 결과를 나타낸 것이다.   
     
   double형을 사용하여 n = 2000일 때의 계산 결과를 나타낸 것이다.  
   var1과 var2 계산 결과가 다르고 var1 방법보다 var2 방법이 계산 시간이 더욱 빠른 것을 알 수 있다. 부동 소수점 연산을 할 때 비슷한 두 수의 뺄셈이 있으면 오차가 발생할 가능성이 크기 때문에 데이터에서 평균을 뺄 때 오차가 발생할 수 있는 1번 방법보다는 계산 순서를 바꿔서 제곱의 덧셈을 먼저 하여 뺄셈 연산을 할 두 수의 차이를 크게 한 2번 방법이 더 정확하다.  
     
   n = 100000일 때 double 자료형을 사용하여 계산하였을 때의 결과이다. 이 때 input 값을 보면 모든 값이 100000인데, 이는 float 자료형으로 소수점을 표현하기 힘들만큼 정수부분이 크기 때문이다. 따라서 이 경우에는 input 값들의 차이가 없으므로 분산이 0이 나오게 된다. 그런데 1번 방법의 경우 분산이 15가 나오는 것으로 보아 1번 방법에서 오차가 많이 발생하는 것을 추측할 수 있다.  
   즉, n이 적당히 큰 수일 때, 부동 소수점의 오차를 최소화할 수 있도록 2번 방법을 쓰는 것이 합리적이다.
2. 과제2  
   2차 방정식이 으로 나타날 때 f(x)의 근을 구할 수 있는 근의 공식은 다음과 같다.   
   이 때 b에 비해 a와 c가 매우 작다면 4ac가 0에 가까워지기 때문에 값이 b에 근사한다. 그러면 에서 부동 소수점의 비슷한 수의 뺄셈 연산으로 인한 오차가 발생할 수 있기 때문에 해당 식을 으로 적절히 변형시켜 주어야한다.  
     
   1) b가 0보다 크거나 같을 때  
   -b가 음수이기 때문에 앞의 부호가 +이면 비슷한 수의 뺄셈 연산으로 인한 오차가 발생할 수 있다. 그래서 b가 0보다 크거나 같은 경우에는 서로 다른 두 근 중 더 큰 근을 구할 때의 근의 공식을 변형해야 한다. 근의 공식의 분모와 분자에 각각 (-b-sqrt(b^2-4ac))을 곱해서 비슷한 수 끼리의 뺄셈 연산을 덧셈으로 바꾼다.  
     
   그러면 위와 같이 b가 양수이고 b에 비해 a와 c가 작은 수일 때의 오차를 최소화할 수 있다. Navie ans는 두 근 중 더 큰 근을 구할 때 오차가 발생하지만 advanced ans는 두 근 모두 오차가 0인 것으로 더욱 정확한 확인할 수 있다.  
     
   2) b가 0보다 작을 때  
   -b가 양수이기 때문에 앞의 부호가 -이면 비슷한 수의 뺄셈 연산으로 인한 오차가 발생할 수 있다. 그래서 b가 0보다 작은 경우에는 서로 다른 두 근 중 더 작은 근을 구할 때의 근의 공식을 변형해야 한다. 근의 공식의 분모와 분자에 각각 (-b+sqrt(b^2-4ac))을 곱해서 비슷한 수 끼리의 뺄셈 연산을 덧셈으로 바꾼다.  
     
   그러면 위와 같이 b가 음수이고 b의 절댓값에 비해 a와 c가 작은 수일 때의 오차를 최소화할 수 있다. Navie ans는 두 근 중 더 작은 근을 구할 때 오차가 발생하지만 advanced ans는 두 근 모두 오차가 0으로 더욱 정확한 것을 확인할 수 있다.
3. 과제3  
   Kahan summation algorithm의 pseudo code는 아래와 같다.  
     
   프로그램에서 input은 rand()함수로 생성된 난수들인데, 일반적인 방법으로 덧셈 연산을 수행하면 sum 변수는 계속 값이 더해져서 아주 큰 수가 되므로 부동 소수점 연산 중 아주 큰 수와 아주 작은 수의 덧셈으로 인한 오차가 발생할 가능성이 있다. 이를 보정하기 위한 방법이 Kahan summation algorithm이다. 이 방법은 덧셈 계산 시 발생하는 오차를 변수 c에 저장했다가 다음 덧셈 계산에서 그 c를 이용하여 보정하는 과정을 반복해서 전체적인 오차를 줄여주는 알고리즘이다. 일반적인 덧셈에 비해 한번 덧셈하는 과정을 4배 정도 늘리기 때문에 연산 시간은 일반적인 덧셈보다 느리지만 부동 소수점의 오차를 보정하는 방법이기 때문에 정확도는 더 높은 알고리즘이다.  
     
   위에서부터 차례로 float형의 일반 덧셈, double형의 일반 덧셈, float형의 Kahan 덧셈, double형의 Kahan 덧셈의 결과이다. float형은 double형 변수보다 표현 가능한 범위가 적기 때문에 double에 비해 오차가 비교적 크다. double형을 사용했을 때는 오차가 거의 없어서 일반 덧셈과 Kahan sum에서 차이가 없다. 그러나 float형을 사용했을 때는 표현 범위가 작아서 normal sum과 Kahan sum의 차이가 비교적 크다. 정확도는 float normal sum < float Kahan sum < double normal sum <= double Kahan sum 순이다. 실험 결과에서도 float형의 일반 덧셈은 다른 값들에 비해 차이가 큰 것을 확인할 수 있다. Float 형의 Kahan sum은 data의 손실이 있긴 하지만 오차를 최소화하여 normal sum에 비해서 비교적 많이 정확하다.   
   Float normal sum의 장점은 변수의 표현 범위가 좁기 때문에 메모리 사용양을 줄일 수 있고 계산 과정이 간단하기 때문에 연산 시간이 빠르다. 단점은 오차로 인한 데이터 손실이 다른 방법에 비해서 많이 크다.  
   double normal sum의 장점은 Kahan sum에 비해 연산 시간이 빠르고 float normal sum에 비해 오차가 작다. 단점은 데이터 표현형이 커서 메모리 사용양이 크다.  
   float Kahan sum의 장점은 변수의 표현 범위가 좁아서 메모리 사용양을 줄일 수 있고 오차를 보정하여 float normal sum에 비해 정확하다. 단점은 double형 보다 float형의 오차가 커서 보정해도 오차가 존재하고 연산이 길어서 연산 시간이 길다.  
   double Kahan sum의 장점은 double의 표현 범위가 넓어서 오차가 적고 오차를 보정하여 오차를 제일 작게할 수 있는 가장 정확한 방법이다. 단점은 데이터형의 메모리 사용양이 크고 연산 시간이 normal sum에 비해 오래걸린다.