*Notes 10/25*

Temporary variables review:

Static\_cast<type>(name) declares a temporary variable of expressed type with the name declared.

When you declare an array, the number of elements that will be in an array must be known at compile time – you can’t declare an array with a number of variables equal to an input. This does not work:

*Int limit;  
cin >> limit;  
int scores [limit] //ERROR*

The best you can do, really, is just to take a really big number for your array limit.

Sample problem:

*Const int MAX\_NUMBER\_SCORES = 10000;  
Int scores [MAX\_NUMBER\_SCORES]; //since it is uninitialized, all values are NOT ZERO – they are random!  
int nscores = 0;//use this int to keep track of how many array spaces are actually being used/which is next  
int total = 0;  
cout << “Enter the scores (negative to quit): “ << endl;  
for (;;)  
{  
 int s;  
 cin >> s;  
 if (s < 0)  
 break;  
 if (nScores == MAX\_NUMBER\_SCORES) //undefined behavior check  
 {  
 cout << “I can handle only ” << MAX\_NUMBER\_SCORES << “ scores!” << endl;  
 cout << “Continuing with just the first “ << MAX\_NUMBER\_SCORES << “ scores.” << endl;  
 }  
 total += s;  
 scores[nScores] = s;  
 nscores++;  
} //This loop continually asks the user for a score until the input is negative or it reaches 10000 scores  
if (nscores == 0)  
 cout << “There were no scores, so no statistics.” << endl;  
else*

*Double mean = static\_cast<double>(total)/nScores  
 cout << “The mean of all the scores is “ << mean << endl;  
 double sumOfSquares = 0;  
 for (int k = 0; k < nScores; k++)  
 {  
 double diff = scores[k] – mean;  
 sumOfSquares += diff \* diff;  
 }  
 cout << “The std. deviation is “ << sqrt(sumOfSquares/nScores) << endl;*

Arrays don’t get passed by value when you call functions like ints or chars. This is notable because it doesn’t pass the actual values in the array or any information about the size of the array.

Here’s an example of how you could do it:

*Double mean(int a[], int n);  
void clear(int a[], int n);  
  
int main()  
{  
 const int MAX\_NUMBER\_SCORES = 10000;  
 int scores[MAX\_NUMBER\_SCORES];  
 int nScores = 0;  
 … //fill up scores array (perhaps partially)  
 double m = mean(scores, nScores);  
 …  
 int stuff[100];  
 …  
 double m2 = mean(stuff, 100);  
 …  
}  
  
double mean(int a[], int n)  
{  
 int total = 0;  
 for (int k = 0; k < n; k++)  
 total += a[k];  
 return static\_cast<double>(total) / n;  
}  
  
void clear(int a[], int n)  
{  
 for (int k = 0; k < n; k++)  
 a[k] = 0;  
}*“Overloading” functions is when you assign 2 different functions to the same name. It’s ok as long as they have different parameters – clear(n, v) and clear(n) is fine because the compiler automatically figures out what you want based on the parameters. Clear(n) and clear(v) would still work as long as n and v are different data types. The only time the compiler gets mad is if n and v are the same data type.