# STAT40780 Data Programming with C (online)

# Lab Sheet 5 (Solutions)

Dr Marie Galligan

Summer 2015

This week's lab requires you to pass arrays from R to C++ through the .C interface, and return arguments back to R indirectly through input arguments to .C. This lab sheet will put into practice the lecture material from week 4 on arrays and control structures in C++.

## 1 Evaluating a relational expression

Write a C++ function (callable from R through the .C interface) that receives a numeric vector from R, squares its elements and returns the vector of squared elements to R. You should loop through the elements of the array using a for loop. Compile this function and call it from R through the .C interface.

### A possible solution

Shown here is a possible solution to this exercise (squareVec function). The squareVec function receives as input **x** (a pointer to a double) and **len** (a pointer to an int). **x** will point to the first element of a copy of the numeric vector passed from R.

The pointer **x** can be subscripted and essentially works as a C++ array, for the purposes of these calculations. By subscripting **x** the elements of the (copy of the) R vector can be modified. Hence, the squared elements will be returned back to R through the input argument **x**. It would also be possible to create a third input argument through which the squared elements of the vector could be passed, but the function below provides a tidier solution.

As C++ will not know the length of the vector passed from R, the length must be passed as an argument to the C++ function also. The pointer **len** points to the length of the array passed from R, and dereferencing this pointer ***len** accesses the integer value respresenting the length.

On the first line of the function body, the pointer **len** is dereferenced and the length of the vector is assigned to the int n.

Next comes a for loop to iterate over the elements of the array and the elements of the array are modified (the values are squared). The squared array is returned back to R through the modified input argument **x**.

**squareVec.cpp**

```


extern "C" {

  void squareVec( double * x   , int * len)
  {

    int n = *len; //length of R vector

    for( int i = 0; i < n ; i++ )
    {
      x[ i ] = x[ i ] * x[ i ];
    }
  }

}
```

Compile the squareVec function and call it from R.

Call squareVec() from R

```
#change working directory to folder where compiled file is stored
setwd("path/to/file")
list.files() #to check for .so of .dll file

dyn.load("squareVec.so") //on OS X
dyn.load("squareVec.dll") //on Windows

x <- c(0, 1, 2, 3)
.C("squareVec", x = as.numeric(x), len = as.integer(length(x)))

dyn.unload("squareVec.so") //on OS X
dyn.unload("squareVec.dll") //on Windows
```

## 2 Calculate the minimum value

Write a C++ function (callable through the .C interface in R) that accepts as input a numeric vector from R, iterates over its elements, and returns the minimum value to R.

## A possible solution

**minval.cpp**

```cpp


extern "C" {

  void minval( double * x  , int * len, double * minval)
  {

    int n = *len; //length of R vector
    *minval = x[ 0 ]; //set minval to first element of x

    for( int i = 1; i < n ; i++ ) //iterate over 2nd element to nth element
    {

       if( x[ i ] < *minval ) //if element indexed by i is less than *minval
       {
         *minval = x[ i ]; //set x[ i ] as current minimum value
       }

    }

  }

}
```

Compile minval.cpp and call from R.

Call minval() from R

```r
#change working directory to folder where compiled file is stored
setwd("path/to/file")
list.files() #to check for .so of .dll file

dyn.load("minval.so") //on OS X
dyn.load("minval.dll") //on Windows

x <- c( 1.0, 0.9, 0.6, 1.2 )
.C("minval", x = as.numeric( x ), len = as.integer(length(x)),
   min = as.numeric(0))

dyn.unload("minval.so") //on OS X
dyn.unload("minval.dll") //on Windows
```

# 3 Using a while loop

Write a C++ function (callable through the .C interface in R) that accepts as input an integer value and computes and returns its factorial to R. Use a while loop in your computation. Call the factorial function from R and compare the output with R's built-in factorial function.

**Solution**

**fact.cpp**

```

extern "C" {

 void fact( int * x, int * fac){

   *fac = 1; //initialize factorial to 1

   while( *x > 0 ){

      *fac *= (*x); //note brackets here not required
      (*x)--; //decrement x

    } // end of while

  } //end of fact

}
```

Compile the factorial function and call it from R:

Calling the compiled factorial function from R

```r
#change working directory to folder where compiled file is stored
setwd("path/to/file")
list.files() #to check for .so of .dll file

dyn.load("fact.so") //on OS X
dyn.load("fact.dll") //on Windows

x <- 10  //integer input argument
.C("fact", as.integer(x), fact = as.integer(1))

#compare result with R's built-in factorial
factorial(x)

dyn.unload("fact.so") //on OS X
dyn.unload("fact.dll") //on Windows
```