Fortran/C/C++ Programming Examples

Prof. Brendan Kochunas

September 2021

1 Hello World

1.1 Fortran

```fortran
PROGRAM hello
  IMPLICIT NONE

  !The ’*’ means default for I/O in Fortran
  WRITE(*,*) "Hello World!"
END PROGRAM
```

Listing 1: Fortran Hello World

1.2 C

```c
#include <stdio.h>

int main() {
   printf("Hello World!");
   return 0;
}
```

Listing 2: C Hello World

1.3 C++

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Listing 3: C++ Hello World

2 Examples with Fixed-Sized Arrays

2.1 Fortran

```fortran
PROGRAM arrays

  USE ISO_FORTRAN_ENV !Defines portable environment and OUTPUT_UNIT parameter

  IMPLICIT NONE

  INTEGER :: i,j,n
  REAL(4) :: spArray(10,10)
  REAL(8) :: dpArray(10,10)

  WRITE(OUTPUT_UNIT,*) "Single precision literal=",0.0
  WRITE(OUTPUT_UNIT,*) "Double precision literal=",0.0d0

  !Fortran natively supports array assignments and elemental arithmetic
  spArray=0.0
  dpArray=0.0d0

  !Initialize values
  n=0
  DO i=1,10
    DO j=1,10
      n=n+1
      spArray(i,j)=REAL(n,4)
      dpArray(i,j)=REAL(n,8)
    ENDDO
  ENDDO

  WRITE(OUTPUT_UNIT,*) spArray(:,1) !First row or column?
  WRITE(OUTPUT_UNIT,*) spArray(1,:) !First row or column?
  WRITE(OUTPUT_UNIT,'(10(f4.1,1x))') dpArray(1:10:2,1) !Array slicing with tuples and formatted output

END PROGRAM
```

Listing 4: Fortran Fixed Arrays

2.2 C

```c
#include<stdio.h>

int main() {

   int i,j,n;
   float  spArray[10][10];
   double dpArray[10][10];

   printf("Single Precision Literal %f\n",0.0f);
   printf("Double Precision Literal %g\n",0.0);

   printf("Check the size! sizeof(0.0f)=%d\n",sizeof(0.0f));
   printf("Check the size! sizeof(0.0)=%d\n",sizeof(0.0));

   //Can you do element-wise assignment?

   n=0;
   for (i=0; i<10; i++) {
      for (j=0; j<10; j++) {
         n++;
         spArray[i][j] = (float)  n;
         dpArray[i][j] = (double) n;
      }
   }

   //First row or column?
   for (i=0; i<10; i++) {
      printf("%f\n",spArray[i][0]);
   }

   //First row or column?
   for (j=0; j<10; j++) {
      printf("%f\n",spArray[0][j]);
   }

   /*
   Can you do array slicing in C?
   */

   return 0;
}
```

Listing 5: C Fixed Arrays

2.3 C++

```cpp
#include<iostream>
#include<valarray>

using namespace std;

int main() {

    // No native multi-dimensional arrays, so we make array of arrays :(
    valarray<valarray<float>  > spArray(valarray<float>( 10),10);
    valarray<valarray<double> > dpArray(valarray<double>(10),10);

    /*
    Can you figure out how to do element-wise assignment?
    */


    int n=0;
    for (int i=0; i<10; i++) {
        for (int j=0; j<10; j++) {
            n++;
            spArray[i][j] = (float)  n;
            dpArray[i][j] = (double) n;
        }
    }

    // Row or column?
    for (int i=0; i<10; i++) {
        cout << spArray[i][0] << "\n";
    }

    // Row or column?
    for (int j=0; j<10; j++) {
        cout << spArray[0][j] << "\n";
    }

    /*
    Can you figure out how to do a slice?
    */

    return 0;
}
```

Listing 6: C++ Basic Arrays

2.4 C++ Templates!

```cpp
#include<iostream>
#include<valarray>
#include<stdexcept>

using namespace std;

template <class T>
class Matrix
{
    public:
        Matrix(size_t rows, size_t cols);
        ~Matrix();
        T& operator() (size_t i, size_t j);
        T  operator() (size_t i, size_t j) const;

    private:
        size_t rows_;
        size_t cols_;
        T *data_;
};

//Define constructor
template <class T>
inline Matrix<T>::Matrix(size_t rows, size_t cols) : rows_(rows), cols_(cols)
{
    if (rows == 0 || cols == 0)
        throw runtime_error("valMatrix cannot be initialized with a 0 dimension!");

    data_ = new T [rows * cols];
}

//Define destructor
template <class T>
inline Matrix<T>::~Matrix()
{
    delete[] data_;
}

//Define &operator
template <class T>
inline T& Matrix<T>::operator() (size_t i, size_t j)
{
    if (i >= rows_ || j >= cols_)
        throw range_error("i or j is out of bounds!");

    return data_[i * cols_ + j];
}

//Define operator
template <class T>
inline T Matrix<T>::operator() (size_t i, size_t j) const
{
    if (i >= rows_ || j >= cols_)
```

```
        throw range_error("i or j is out of bounds!");

    return data_[i * cols_ + j];
}


int main() {

    // No native multi-dimensional arrays, so we make our own templated class
    Matrix<float>  spArray(10,10);
    Matrix<double> dpArray(10,10);

    /*
    Can you figure out how to do element-wise assignment?
    */


    int n=0;
    for (int i=0; i<10; i++) {
        for (int j=0; j<10; j++) {
            n++;
            spArray(i,j) =  (float) n;
            dpArray(i,j) = (double) n;
        }
    }

    // Row or column?
    for (int i=0; i<10; i++) {
        cout << spArray(i,0) << "\n";
    }

    // Row or column?
    for (int j=0; j<10; j++) {
        cout << spArray(0,j) << "\n";
    }

    /*
    Can you figure out how to do a slice?
    */
    return 0;
}
```

Listing 7: C++ Templated Arrays

3 Examples with Dynamically-Sized Arrays

3.1 Fortran

```fortran
PROGRAM arrays

  USE ISO_FORTRAN_ENV !Defines portable environment and OUTPUT_UNIT parameter

  IMPLICIT NONE

  INTEGER :: i,j,n
  REAL(4),ALLOCATABLE :: spArray(:,:) !basic allocatable array
  REAL(8),POINTER :: ptrArray(:,:),dpArray(:,:) !In Fortran pointers are strongly typed

  !Allocate memory
  WRITE(*,*) ALLOCATED(spArray)
  WRITE(*,*) ASSOCIATED(dpArray) !The return value here is actually undefined.
  ALLOCATE(spArray(10,10)) !Allocates a block of memory for 400 bytes
  ALLOCATE(dpArray(10,10)) !Allocates a block of memory for 800 bytes
  WRITE(*,*) ALLOCATED(spArray)
  WRITE(*,*) ASSOCIATED(dpArray) !Associated is used for variables with POINTER attribute, not ALLOCATABLE

  !Fortran natively supports array assignments and elemental arithmetic
  spArray=0.0
  dpArray=0.0d0
  ptrArray => NULL()

  !Pointer Assignment and association
  WRITE(*,*) ASSOCIATED(ptArray)
  ptrArray => dpArray
  WRITE(*,*) ASSOCIATED(ptArray)
  !ptrArray => spArray this would produce a compile-time error

  !Initialize values
  n=0
  DO i=1,10
    DO j=1,10
      n=n+1
      spArray(i,j)=REAL(n,4)
      dpArray(i,j)=REAL(n,8)
    ENDDO
  ENDDO

  WRITE(OUTPUT_UNIT,*) spArray(:,1) !First row or column?
  WRITE(OUTPUT_UNIT,*) spArray(1,:) !First row or column?
  WRITE(OUTPUT_UNIT,'(10(f4.1,1x))') dpArray(1:10:2,1) !Array slicing with tuples and formatted output
  WRITE(OUTPUT_UNIT,'(10(f4.1,1x))') ptrArray(1:10:2,1) !Pointer points to memory associated with dpArray

  !Deallocation of memory
  DEALLOCATE(spAarry) !Allocatable variables are automatically deallocated when out of scope, so this is not necessary
  DEALLOCATE(dpArray) !Pointers must be deallocated before they leave scope, otherwise it is a memory leak

  !if elements of ptrArray are accessed here it is a segfault
  WRITE(*,*) ASSOCIATED(ptrArray)
  ptrArray => NULL()  !dissociates ptrArray from location of memory assigned to dpArray--does not modify memory usage


END PROGRAM
```

Listing 8: Fortran Dynamic Arrays

3.2 C

```c
#include<stdio.h>
#include<stdlib.h>

int main() {

   int i,j,n;
   int N;
   float**   spArray;
   double**  dpArray;

   printf("Check the size: sizeof(0.0f)=%d\n",sizeof(0.0f));
   printf("Check the size: sizeof(0.0)=%d\n",sizeof(0.0));
   printf("Check the size: sizeof(*spArray)=%d\n",sizeof(*spArray));
   printf("Check the size: sizeof(*dpArray)=%d\n",sizeof(*dpArray));

   N=10;
   spArray = malloc(N*sizeof(*spArray)); //Allocate rows
   dpArray = malloc(N*sizeof(*dpArray));
   for (i=0; i<N; i++) {
     spArray[i] = malloc(N*sizeof(*spArray[i]));
     dpArray[i] = malloc(N*sizeof(*dpArray[i]));
   }

   n=0;
   for (i=0; i<10; i++) {
     for (j=0; j<10; j++) {
       n++;
       spArray[i][j] = (float)  n; //In fortran REAL(n,4)
       dpArray[i][j] = (double) n;
     }
   }

   //Print first row or column?
   for (i=0; i<10; i++) {
     printf("%f\n",spArray[i][0]);
   }
   for (j=0; j<10; j++) {
     printf("%f\n",dpArray[0][j]);
   }

   //Do not forget to delete your arrays from memory!
   for (i=0; i<N; i++) {
     free(spArray[i]);
     free(dpArray[i]);
   }
   free(spArray);
   free(dpArray);

   return 0;
}
```

Listing 9: C Dynamic Arrays