Fortran/C/C++ Programming Examples

Prof. Brendan Kochunas

September 2021

1 Hello World

1.1 Fortran

```
PROGRAM hello
IMPLICIT NONE

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

Listing 1: Fortran Hello World

1.2 C

```
#include <stdio.h>

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

Listing 2: C Hello World

1.3 C++

```
#include <iostream>

int main() {

std::cout << "Hello World!";

return 0;
}</pre>
```

Listing 3: C++ Hello World

2 Examples with Fixed-Sized Arrays

2.1 Fortran

```
1 PROGRAM arrays
    USE ISO_FORTRAN_ENV !Defines portable environment and OUTPUT_UNIT
       parameter
    IMPLICIT NONE
5
    INTEGER :: i,j,n
    REAL(4) :: spArray(10,10)
REAL(8) :: dpArray(10,10)
8
10
    WRITE(OUTPUT_UNIT,*) "Single precision literal=",0.0
11
    WRITE(OUTPUT_UNIT,*) "Double precision literal=",0.0d0
12
13
     !Fortran natively supports array assignments and elemental
14
      arithmetic
15
    spArray=0.0
    dpArray=0.0d0
16
17
    !Initialize values
18
19
20
    DO i=1,10
      DO j=1,10
21
22
        spArray(i,j)=REAL(n,4)
23
        dpArray(i,j)=REAL(n,8)
24
      ENDDO
25
    ENDDO
26
    WRITE(OUTPUT_UNIT,*) spArray(:,1) !First row or column?
28
    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
32 END PROGRAM
```

Listing 4: Fortran Fixed Arrays

2.2 C

```
#include < stdio.h>
3 int main() {
     int i,j,n;
float spArray[10][10];
double dpArray[10][10];
6
8
      printf("Single Precision Literal %f\n",0.0f);
9
      printf("Double Precision Literal %g\n",0.0);
10
11
12
      printf("Check the size! sizeof(0.0f)=\frac{1}{2}d\ln, sizeof(0.0f));
      printf("Check the size! sizeof(0.0)=%d\n", sizeof(0.0));
13
14
      //Can you do element-wise assignment?
15
16
17
      n=0;
      for (i=0; i<10; i++) {</pre>
18
         for (j=0; j<10; j++) {</pre>
19
20
             n++;
             spArray[i][j] = (float) n;
21
             dpArray[i][j] = (double) n;
22
         }
23
24
25
      //First row or column? for (i=0; i<10; i++) {
26
27
         printf("%f\n",spArray[i][0]);
28
29
30
31
      //First row or column?
      for (j=0; j<10; j++) {</pre>
32
         printf("%f\n",spArray[0][j]);
33
34
35
36
      Can you do array slicing in C?
37
      */
38
39
      return 0;
40
41 }
```

Listing 5: C Fixed Arrays

2.3 C++

```
#include < iostream >
#include < valarray >
4 using namespace std;
6 int main() {
      // No native multi-dimensional arrays, so we make array of
      arrays :(
      valarray <valarray <double > > dpArray(valarray <double > (10),10);
10
11
12
      Can you figure out how to do element-wise assignment?
13
14
      */
15
16
      int n=0;
17
      for (int i=0; i<10; i++) {</pre>
18
          for (int j=0; j<10; j++) {</pre>
19
20
              n++;
              spArray[i][j] = (float) n;
21
               dpArray[i][j] = (double) n;
22
          }
23
      }
24
25
26
      // Row or column?
      for (int i=0; i<10; i++) {</pre>
27
          cout << spArray[i][0] << "\n";</pre>
28
29
30
      // Row or column?
31
      for (int j=0; j<10; j++) {</pre>
32
          cout << spArray[0][j] << "\n";</pre>
33
34
35
36
37
      Can you figure out how to do a slice?
38
39
40
      return 0;
41 }
```

Listing 6: C++ Basic Arrays

2.4 C++ Templates!

```
#include < iostream >
#include < valarray >
3 #include < stdexcept >
s using namespace std;
7 template <class T>
8 class Matrix
9 {
10
      public:
          Matrix(size_t rows, size_t cols);
11
          ~Matrix();
12
          T& operator() (size_t i, size_t j);
13
          T operator() (size_t i, size_t j) const;
14
15
      private:
16
17
          size_t rows_;
          size_t cols_;
18
          T *data_;
19
20 };
21
22 //Define constructor
23 template <class T>
24 inline Matrix<T>::Matrix(size_t rows, size_t cols) : rows_(rows),
      cols_(cols)
25 {
      if (rows == 0 || cols == 0)
26
          throw runtime_error("valMatrix cannot be initialized with a
27
       0 dimension!");
28
29
      data_ = new T [rows * cols];
30 }
31
32 //Define destructor
33 template <class T>
34 inline Matrix <T>:: "Matrix()
35 {
      delete[] data_;
36
37 }
38
39 //Define &operator
40 template <class T>
inline T& Matrix<T>::operator() (size_t i, size_t j)
42 {
      if (i >= rows_ || j >= cols_)
43
44
          throw range_error("i or j is out of bounds!");
45
46
      return data_[i * cols_ + j];
47 }
49 //Define operator
50 template <class T>
51 inline T Matrix<T>::operator() (size_t i, size_t j) const
52 {
if (i >= rows_ || j >= cols_)
```

```
throw range_error("i or j is out of bounds!");
54
55
       return data_[i * cols_ + j];
56
57 }
58
59
60 int main() {
61
        // No native multi-dimensional arrays, so we make our own
62
        templated class
       Matrix < float > spArray(10,10);
Matrix < double > dpArray(10,10);
63
64
65
66
       Can you figure out how to do element-wise assignment?
67
68
69
70
71
       int n=0;
       for (int i=0; i<10; i++) {
   for (int j=0; j<10; j++) {</pre>
72
73
                 n++;
74
75
                 spArray(i,j) = (float) n;
                 dpArray(i,j) = (double) n;
76
77
            }
       }
78
79
       // Row or column?
80
       for (int i=0; i<10; i++) {</pre>
81
            cout << spArray(i,0) << "\n";</pre>
82
83
84
        // Row or column?
85
       for (int j=0; j<10; j++) {</pre>
86
            cout << spArray(0,j) << "\n";</pre>
87
88
89
90
       Can you figure out how to do a slice?
91
92
       return 0;
93
94 }
```

Listing 7: C++ Templated Arrays

3 Examples with Dynamically-Sized Arrays

3.1 Fortran

```
1 PROGRAM arrays
    USE ISO_FORTRAN_ENV !Defines portable environment and OUTPUT_UNIT
       parameter
    IMPLICIT NONE
5
    INTEGER :: i,j,n
    REAL(4), ALLOCATABLE :: spArray(:,:) !basic allocatable array
    REAL(8),POINTER :: ptrArray(:,:),dpArray(:,:) !In Fortran
      pointers are strongly typed
10
    ! Allocate memory
11
    WRITE(*,*) ALLOCATED(spArray)
12
13
    WRITE(*,*) ASSOCIATED(dpArray) !The return value here is actually
        undefined.
    ALLOCATE(spArray(10,10)) ! Allocates a block of memory for 400
      bvtes
    ALLOCATE(dpArray(10,10)) !Allocates a block of memory for 800
15
      bytes
    WRITE(*,*) ALLOCATED(spArray)
16
17
    WRITE(*,*) ASSOCIATED(dpArray) ! Associated is used for variables
      with POINTER attribute, not ALLOCATABLE
18
    !Fortran natively supports array assignments and elemental
19
      arithmetic
20
    spArray=0.0
    dpArray=0.0d0
21
22
    ptrArray => NULL()
23
    !Pointer Assignment and association
24
25
    WRITE(*,*) ASSOCIATED(ptArray)
    ptrArray => dpArray
26
27
    WRITE(*,*) ASSOCIATED(ptArray)
    !ptrArray => spArray this would produce a compile-time error
28
29
    !Initialize values
30
    n=0
31
    D0 i=1,10
32
      D0 j=1,10
33
        n=n+1
34
        spArray(i,j)=REAL(n,4)
35
36
        dpArray(i,j)=REAL(n,8)
      ENDDO
37
    ENDDO
38
    WRITE(OUTPUT_UNIT,*) spArray(:,1) !First row or column?
40
     WRITE(OUTPUT_UNIT,*) spArray(1,:) !First row or column?
41
    WRITE(OUTPUT_UNIT, '(10(f4.1,1x))') dpArray(1:10:2,1) !Array
42
      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
45
    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
48
    !if elements of ptrArray are accessed here it is a segfault
49
    WRITE(*,*) ASSOCIATED(ptrArray)
50
    ptrArray => NULL() !dissociates ptrArray from location of memory
51
       assigned to dpArray--does not modify memory usage
52
53
54 END PROGRAM
```

Listing 8: Fortran Dynamic Arrays

3.2 C

```
#include < stdio.h>
2 #include < stdlib.h>
4 int main() {
      int i,j,n;
6
      int N;
7
      float **
8
                 spArray;
      double** dpArray;
9
10
      printf("Check the size: sizeof(0.0f)=%d\n", sizeof(0.0f));
11
      printf("Check the size: sizeof(0.0) = \frac{1}{2} d n, sizeof(0.0));
12
      printf("Check the size: sizeof(*spArray)=%d\n", sizeof(*spArray))
13
      printf("Check the size: sizeof(*dpArray)=%d\n", sizeof(*dpArray))
      ;
15
      N=10;
16
      spArray = malloc(N*sizeof(*spArray)); //Allocate rows
17
      dpArray = malloc(N*sizeof(*dpArray));
18
      for (i=0; i<N; i++) {</pre>
19
20
        spArray[i] = malloc(N*sizeof(*spArray[i]));
        dpArray[i] = malloc(N*sizeof(*dpArray[i]));
21
22
23
      n=0;
24
25
      for (i=0; i<10; i++) {</pre>
       for (j=0; j<10; j++) {
26
27
          spArray[i][j] = (float) n; //In fortran REAL(n,4)
28
29
          dpArray[i][j] = (double) n;
        }
30
31
32
      //Print first row or column?
33
      for (i=0; i<10; i++) {</pre>
34
       printf("%f\n",spArray[i][0]);
35
36
37
      for (j=0; j<10; j++) {</pre>
        printf("%f\n",dpArray[0][j]);
38
39
40
      //Do not forget to delete your arrays from memory!
41
42
      for (i=0; i<N; i++) {</pre>
        free(spArray[i]);
43
44
        free(dpArray[i]);
45
46
      free(spArray);
      free(dpArray);
47
48
49
      return 0;
50 }
```

Listing 9: C Dynamic Arrays