

Fortran/C/C++ Programming Examples

Prof. Brendan Kochunas

September 2021

1 Hello World

1.1 Fortran

```
1 PROGRAM hello
2   IMPLICIT NONE
3
4   !The '*' means default for I/O in Fortran
5   WRITE(*,*) "Hello World!"
6 END PROGRAM
```

Listing 1: Fortran Hello World

1.2 C

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!");
5     return 0;
6 }
```

Listing 2: C Hello World

1.3 C++

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Hello World!";
5     return 0;
6 }
```

Listing 3: C++ Hello World

2 Examples with Fixed-Sized Arrays

2.1 Fortran

```
1 PROGRAM arrays
2
3   USE ISO_FORTRAN_ENV !Defines portable environment and OUTPUT_UNIT
4     parameter
5
6   IMPLICIT NONE
7
8   INTEGER :: i,j,n
9   REAL(4) :: spArray(10,10)
10  REAL(8) :: dpArray(10,10)
11
12  WRITE(OUTPUT_UNIT,*) "Single precision literal=",0.0
13  WRITE(OUTPUT_UNIT,*) "Double precision literal=",0.0d0
14
15  !Fortran natively supports array assignments and elemental
16    arithmetic
17  spArray=0.0
18  dpArray=0.0d0
19
20  !Initialize values
21  n=0
22  DO i=1,10
23    DO j=1,10
24      n=n+1
25      spArray(i,j)=REAL(n,4)
26      dpArray(i,j)=REAL(n,8)
27    ENDDO
28  ENDDO
29
30  WRITE(OUTPUT_UNIT,*) spArray(:,1) !First row or column?
31  WRITE(OUTPUT_UNIT,*) spArray(1,:) !First row or column?
32  WRITE(OUTPUT_UNIT,'(10(f4.1,1x))') dpArray(1:10:2,1) !Array
33    slicing with tuples and formatted output
34
35  END PROGRAM
```

Listing 4: Fortran Fixed Arrays

2.2 C

```
1 #include<stdio.h>
2
3 int main() {
4
5     int i,j,n;
6     float  spArray[10][10];
7     double dpArray[10][10];
8
9     printf("Single Precision Literal %f\n",0.0f);
10    printf("Double Precision Literal %g\n",0.0);
11
12    printf("Check the size! sizeof(0.0f)=%d\n",sizeof(0.0f));
13    printf("Check the size! sizeof(0.0)=%d\n",sizeof(0.0));
14
15    //Can you do element-wise assignment?
16
17    n=0;
18    for (i=0; i<10; i++) {
19        for (j=0; j<10; j++) {
20            n++;
21            spArray[i][j] = (float)  n;
22            dpArray[i][j] = (double) n;
23        }
24    }
25
26    //First row or column?
27    for (i=0; i<10; i++) {
28        printf("%f\n",spArray[i][0]);
29    }
30
31    //First row or column?
32    for (j=0; j<10; j++) {
33        printf("%f\n",spArray[0][j]);
34    }
35
36    /*
37    Can you do array slicing in C?
38    */
39
40    return 0;
41 }
```

Listing 5: C Fixed Arrays

2.3 C++

```
1 #include<iostream>
2 #include<valarray>
3
4 using namespace std;
5
6 int main() {
7
8     // No native multi-dimensional arrays, so we make array of
9     arrays :(
10    valarray<valarray<float>> > spArray(valarray<float>( 10),10);
11    valarray<valarray<double>> > dpArray(valarray<double>(10),10);
12
13    /*
14    Can you figure out how to do element-wise assignment?
15    */
16
17    int n=0;
18    for (int i=0; i<10; i++) {
19        for (int j=0; j<10; j++) {
20            n++;
21            spArray[i][j] = (float) n;
22            dpArray[i][j] = (double) n;
23        }
24    }
25
26    // Row or column?
27    for (int i=0; i<10; i++) {
28        cout << spArray[i][0] << "\n";
29    }
30
31    // Row or column?
32    for (int j=0; j<10; j++) {
33        cout << spArray[0][j] << "\n";
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!

```
1 #include<iostream>
2 #include<valarray>
3 #include<stdexcept>
4
5 using namespace std;
6
7 template <class T>
8 class Matrix
9 {
10     public:
11         Matrix(size_t rows, size_t cols);
12         ~Matrix();
13         T& operator() (size_t i, size_t j);
14         T operator() (size_t i, size_t j) const;
15
16     private:
17         size_t rows_;
18         size_t cols_;
19         T *data_;
20 };
21
22 //Define constructor
23 template <class T>
24 inline Matrix<T>::Matrix(size_t rows, size_t cols) : rows_(rows),
25     cols_(cols)
26 {
27     if (rows == 0 || cols == 0)
28         throw runtime_error("valMatrix cannot be initialized with a
29             0 dimension!");
30
31     data_ = new T [rows * cols];
32 }
33
34 //Define destructor
35 template <class T>
36 inline Matrix<T>::~~Matrix()
37 {
38     delete[] data_;
39 }
40
41 //Define &operator
42 template <class T>
43 inline T& Matrix<T>::operator() (size_t i, size_t j)
44 {
45     if (i >= rows_ || j >= cols_)
46         throw range_error("i or j is out of bounds!");
47
48     return data_[i * cols_ + j];
49 }
50
51 //Define operator
52 template <class T>
53 inline T Matrix<T>::operator() (size_t i, size_t j) const
54 {
55     if (i >= rows_ || j >= cols_)
```

```

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

```

Listing 7: C++ Templated Arrays

3 Examples with Dynamically-Sized Arrays

3.1 Fortran

```
1 PROGRAM arrays
2
3   USE ISO_FORTRAN_ENV !Defines portable environment and OUTPUT_UNIT
4     parameter
5
6   IMPLICIT NONE
7
8   INTEGER :: i,j,n
9   REAL(4),ALLOCATABLE :: spArray(:,:) !basic allocatable array
10  REAL(8),POINTER :: ptrArray(:,:),dpArray(:,:) !In Fortran
11    pointers are strongly typed
12
13  !Allocate memory
14  WRITE(*,*) ALLOCATED(spArray)
15  WRITE(*,*) ASSOCIATED(dpArray) !The return value here is actually
16    undefined.
17  ALLOCATE(spArray(10,10)) !Allocates a block of memory for 400
18    bytes
19  ALLOCATE(dpArray(10,10)) !Allocates a block of memory for 800
20    bytes
21  WRITE(*,*) ALLOCATED(spArray)
22  WRITE(*,*) ASSOCIATED(dpArray) !Associated is used for variables
23    with POINTER attribute, not ALLOCATABLE
24
25  !Fortran natively supports array assignments and elemental
26    arithmetic
27  spArray=0.0
28  dpArray=0.0d0
29  ptrArray => NULL()
30
31  !Pointer Assignment and association
32  WRITE(*,*) ASSOCIATED(ptrArray)
33  ptrArray => dpArray
34  WRITE(*,*) ASSOCIATED(ptrArray)
35  !ptrArray => spArray this would produce a compile-time error
36
37  !Initialize values
38  n=0
39  DO i=1,10
40    DO j=1,10
41      n=n+1
42      spArray(i,j)=REAL(n,4)
43      dpArray(i,j)=REAL(n,8)
44    ENDDO
45  ENDDO
46
47  WRITE(OUTPUT_UNIT,*) spArray(:,1) !First row or column?
48  WRITE(OUTPUT_UNIT,*) spArray(1,:) !First row or column?
49  WRITE(OUTPUT_UNIT,'(10(f4.1,1x))') dpArray(1:10:2,1) !Array
50    slicing with tuples and formatted output
51  WRITE(OUTPUT_UNIT,'(10(f4.1,1x))') ptrArray(1:10:2,1) !Pointer
52    points to memory associated with dpArray
```

```

45  !Deallocation of memory
46  DEALLOCATE(spAarry) !Allocatable variables are automatically
    deallocated when out of scope, so this is not necessary
47  DEALLOCATE(dpArray) !Pointers must be deallocated before they
    leave scope, otherwise it is a memory leak
48
49  !if elements of ptrArray are accessed here it is a segfault
50  WRITE(*,*) ASSOCIATED(ptrArray)
51  ptrArray => NULL() !dissociates ptrArray from location of memory
    assigned to dpArray--does not modify memory usage
52
53
54  END PROGRAM

```

Listing 8: Fortran Dynamic Arrays

3.2 C

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5
6     int i,j,n;
7     int N;
8     float**  spArray;
9     double** dpArray;
10
11     printf("Check the size: sizeof(0.0f)=%d\n",sizeof(0.0f));
12     printf("Check the size: sizeof(0.0)=%d\n",sizeof(0.0));
13     printf("Check the size: sizeof(*spArray)=%d\n",sizeof(*spArray))
14     ;
15     printf("Check the size: sizeof(*dpArray)=%d\n",sizeof(*dpArray))
16     ;
17
18     N=10;
19     spArray = malloc(N*sizeof(*spArray)); //Allocate rows
20     dpArray = malloc(N*sizeof(*dpArray));
21     for (i=0; i<N; i++) {
22         spArray[i] = malloc(N*sizeof(*spArray[i]));
23         dpArray[i] = malloc(N*sizeof(*dpArray[i]));
24     }
25
26     n=0;
27     for (i=0; i<10; i++) {
28         for (j=0; j<10; j++) {
29             n++;
30             spArray[i][j] = (float)  n; //In fortran REAL(n,4)
31             dpArray[i][j] = (double) n;
32         }
33     }
34
35     //Print first row or column?
36     for (i=0; i<10; i++) {
37         printf("%f\n",spArray[i][0]);
38     }
39     for (j=0; j<10; j++) {
40         printf("%f\n",dpArray[0][j]);
41     }
42
43     //Do not forget to delete your arrays from memory!
44     for (i=0; i<N; i++) {
45         free(spArray[i]);
46         free(dpArray[i]);
47     }
48     free(spArray);
49     free(dpArray);
50
51     return 0;
52 }
```

Listing 9: C Dynamic Arrays