Structured Data Types

Chap 12

- The data types we have considered so far all had a single value:
  - Int
  - Float
  - String (we view strings as immutable)
- Structured data types are typically made up of/contain *multiple values*
  - Arrays
  - Class structures
  - Enums
- Here we will take a look at arrays.

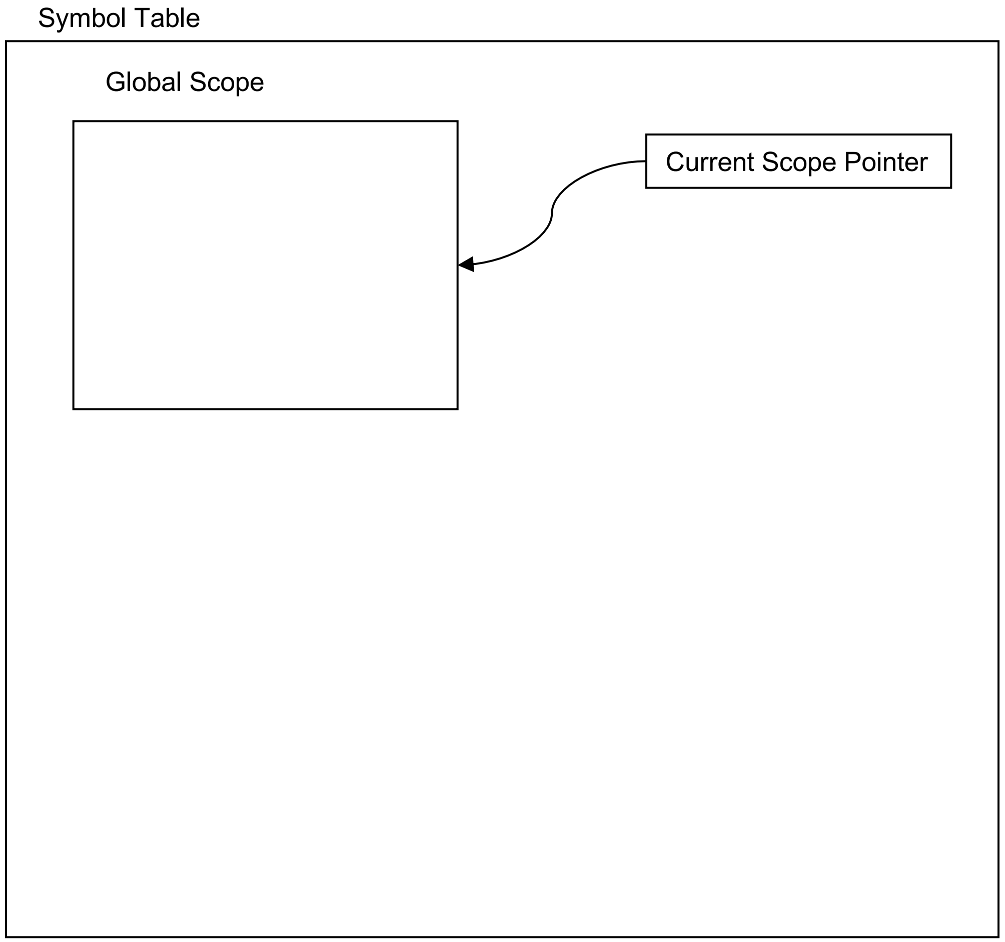
Arrays

- Arrays are data structures that look like lists where every element in the list is of the same data type.
- A convenient way to view arrays is that of a structure that can hold multiple values:
  - int[3] v - v is a (array) variable that holds integer arrays of size three.

Arrays

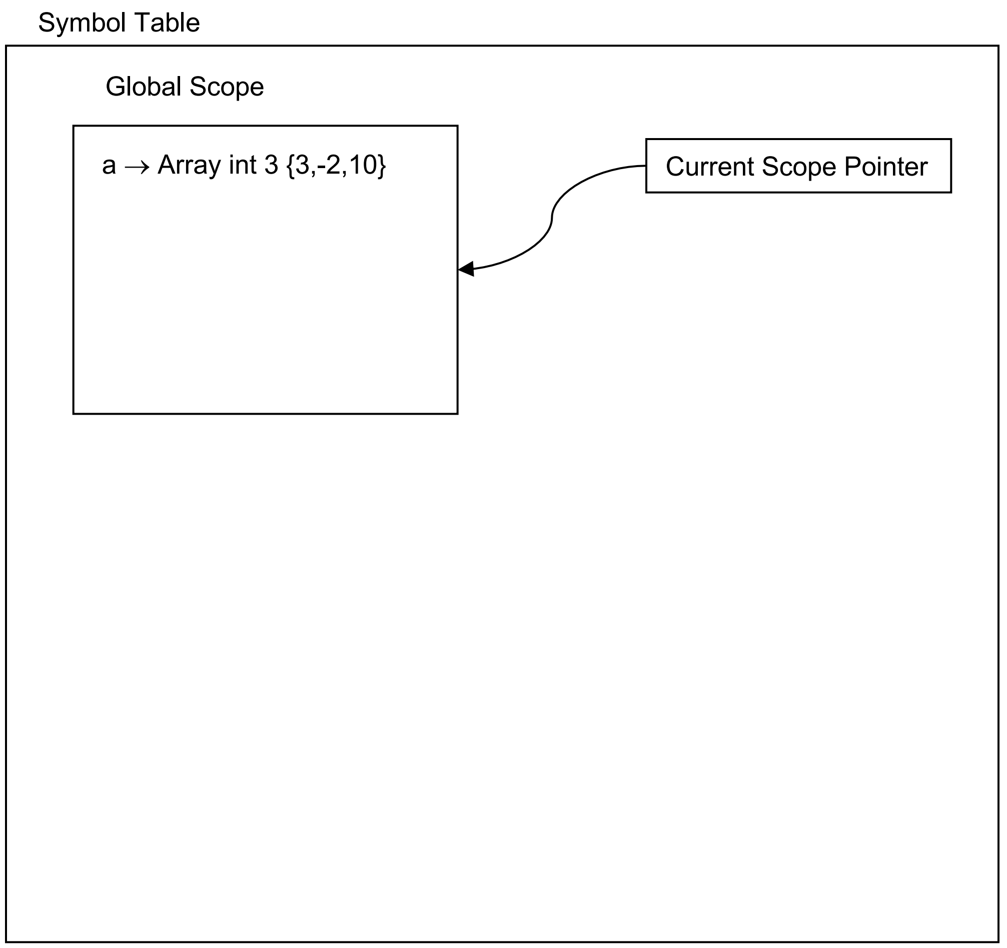
- Initializers
  - int[3] a = { 3,-2,10 };
- Arrays can be viewed as *array values*
  - int[3] a = { 3,-2,10 };
  - int[3] b = a; ← copy values from a to b
- The size of the array and the type of the elements matters
  - int[3] a = { 3,-2,10};
  - float[3] b = a; ✗
  - or
  - int[4] b = a; ✗

Interpreting Arrays



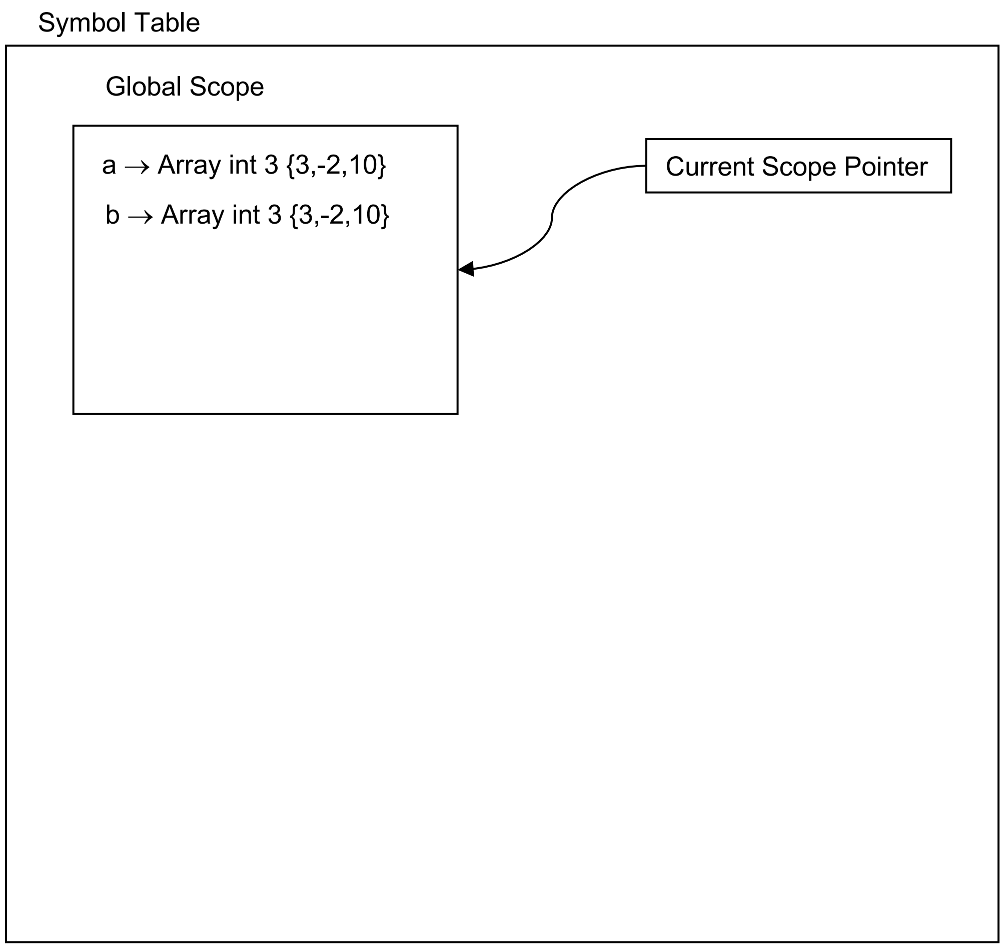
```
int[3] a = { 3,-2,10 };
int[3] b = a;
b[1] = 0;
```

Interpreting Arrays



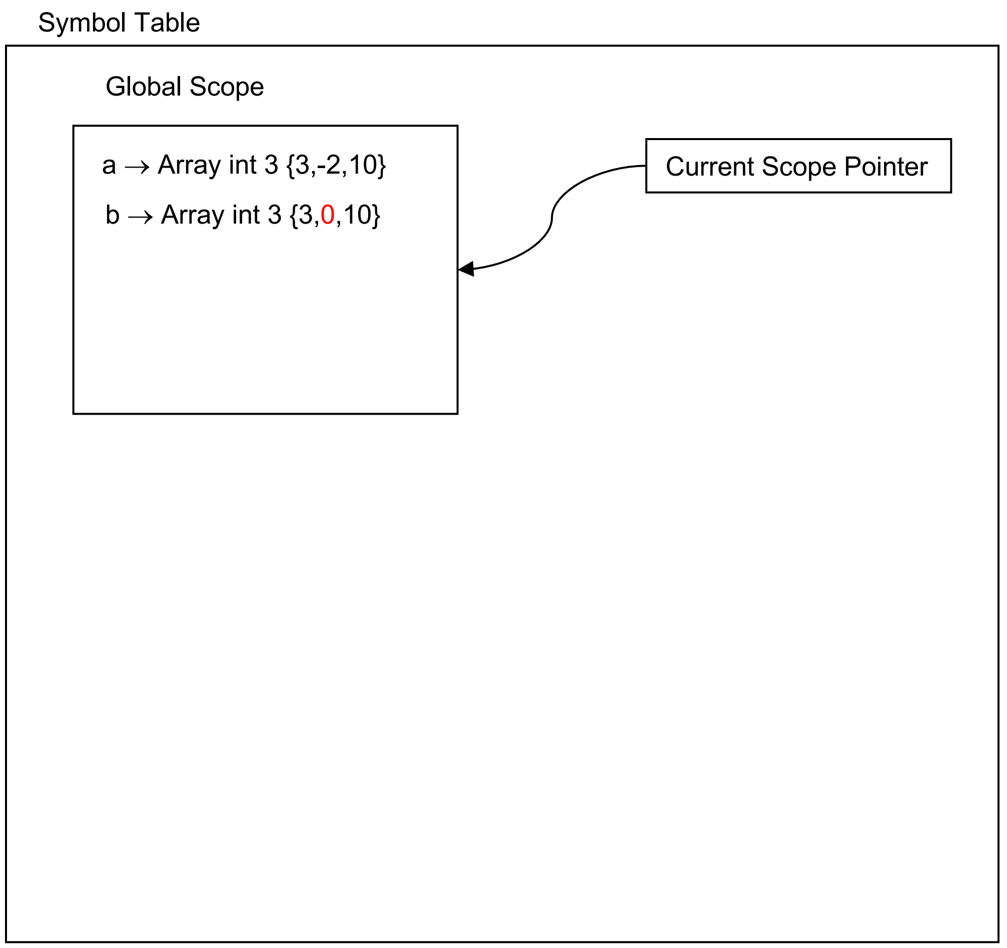
```
int[3] a = { 3,-2,10 };
int[3] b = a;
b[1] = 0;
```

Interpreting Arrays



```
int[3] a = { 3,-2,10 };
int[3] b = a;
b[1] = 0;
```

Interpreting Arrays



```
int[3] a = { 3,-2,10 };
int[3] b = a;
b[1] = 0;
```

Computing with Arrays

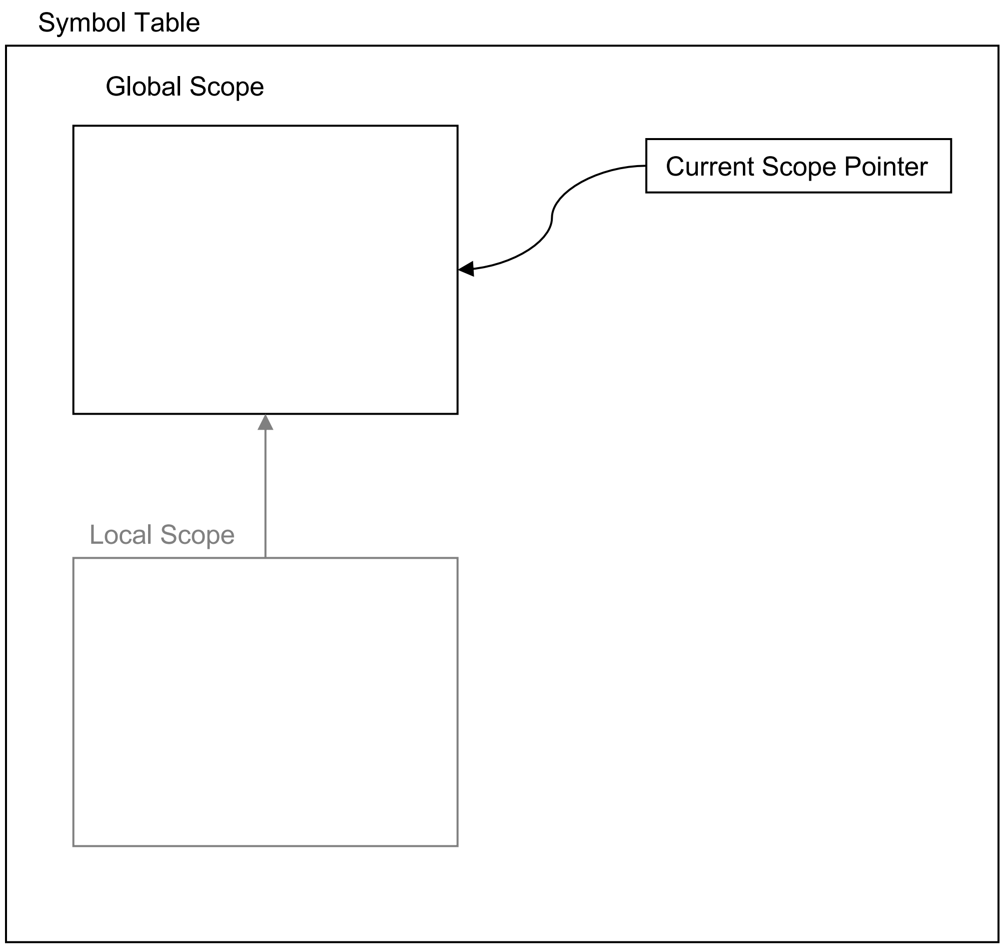
- Just as in the case of scalar variables, array variables can appear in two types of contexts:
  - Expressions: here we read the contents of the array location indexed, e.g., x = a[2].
  - Assignment statements: here we access the index array location and update its contents, e.g., a[2] = x

Computing with Arrays

- Here is a program that computes a sequence of numbers into an array:

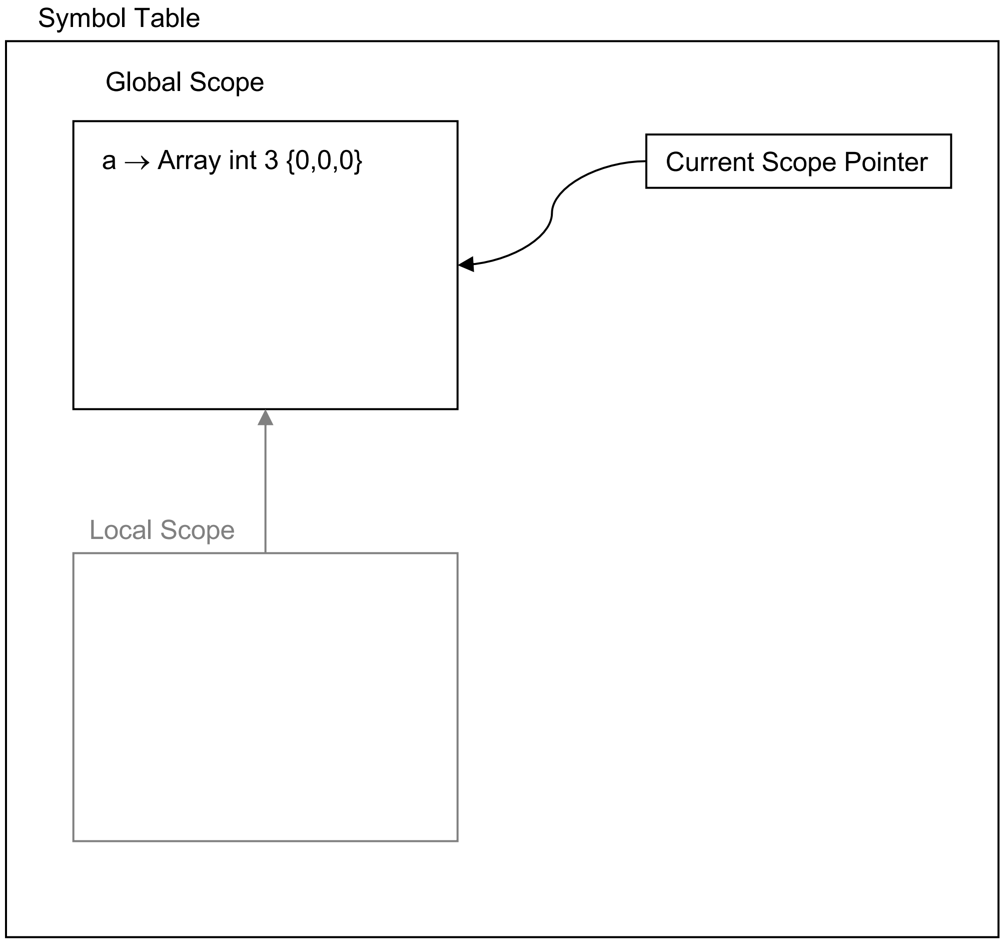
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “, a;
```

Interpreting Arrays



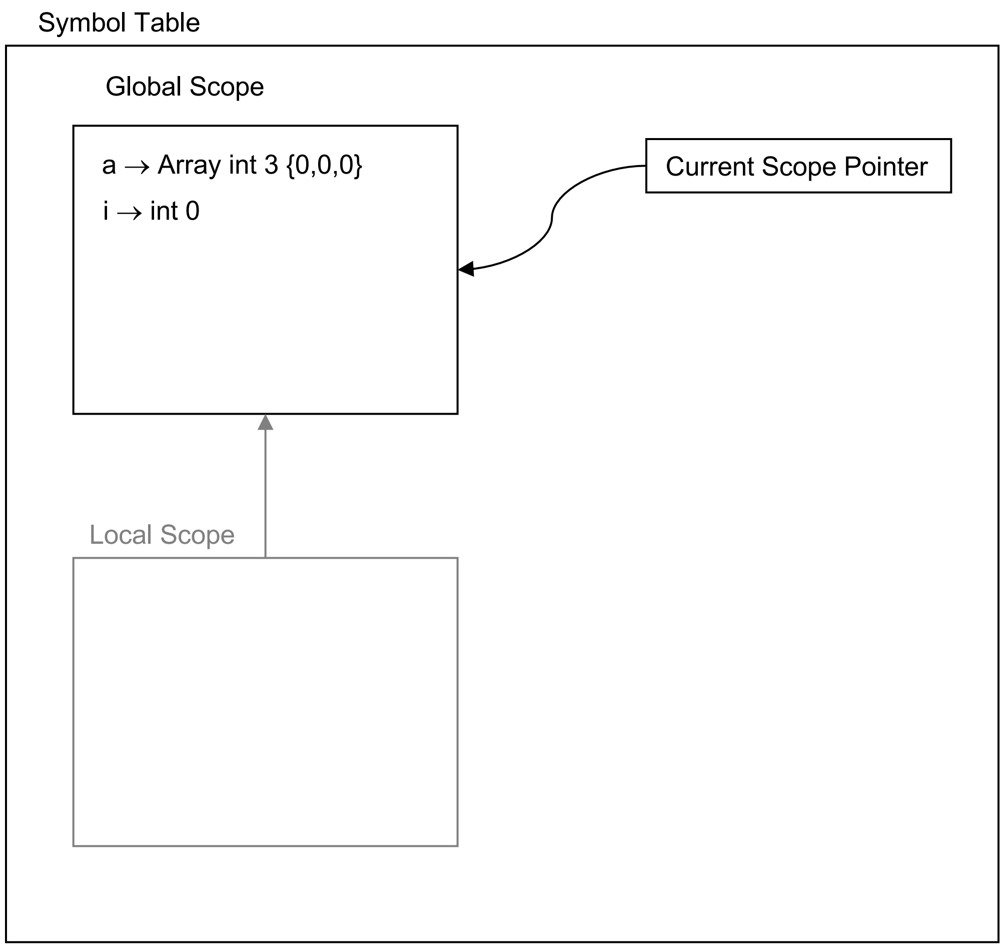
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



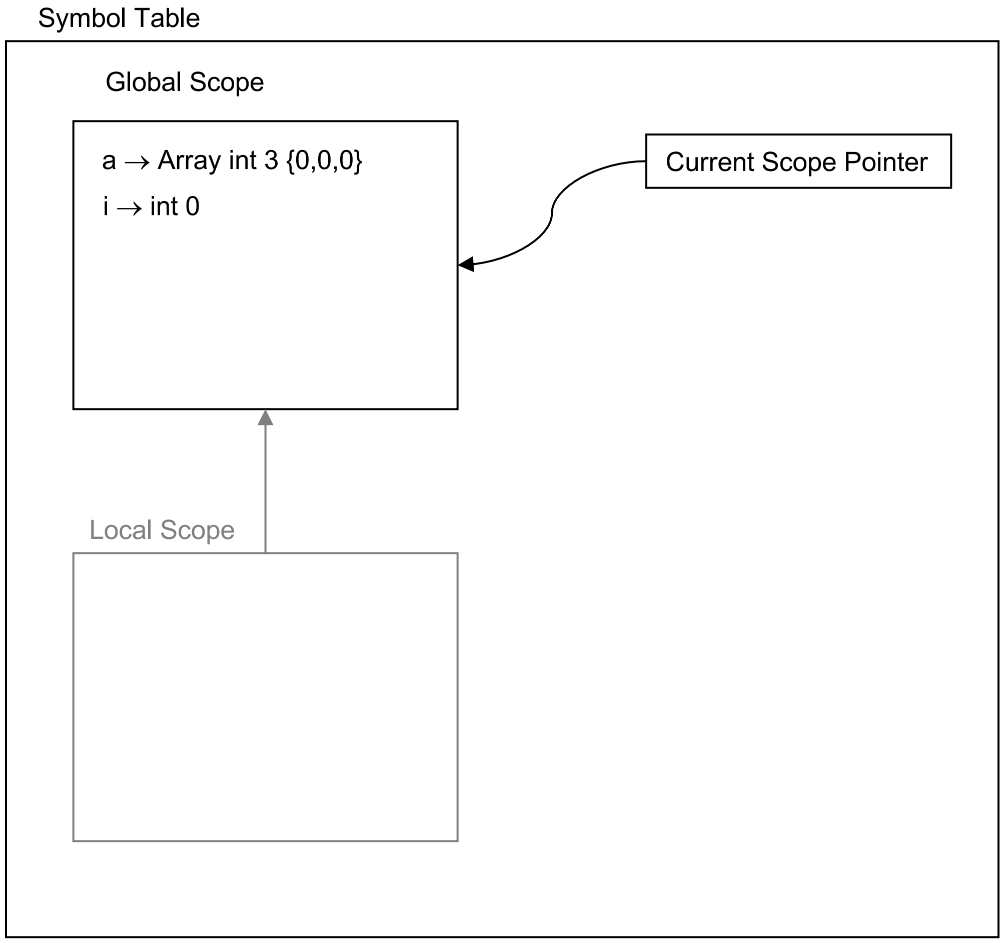
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



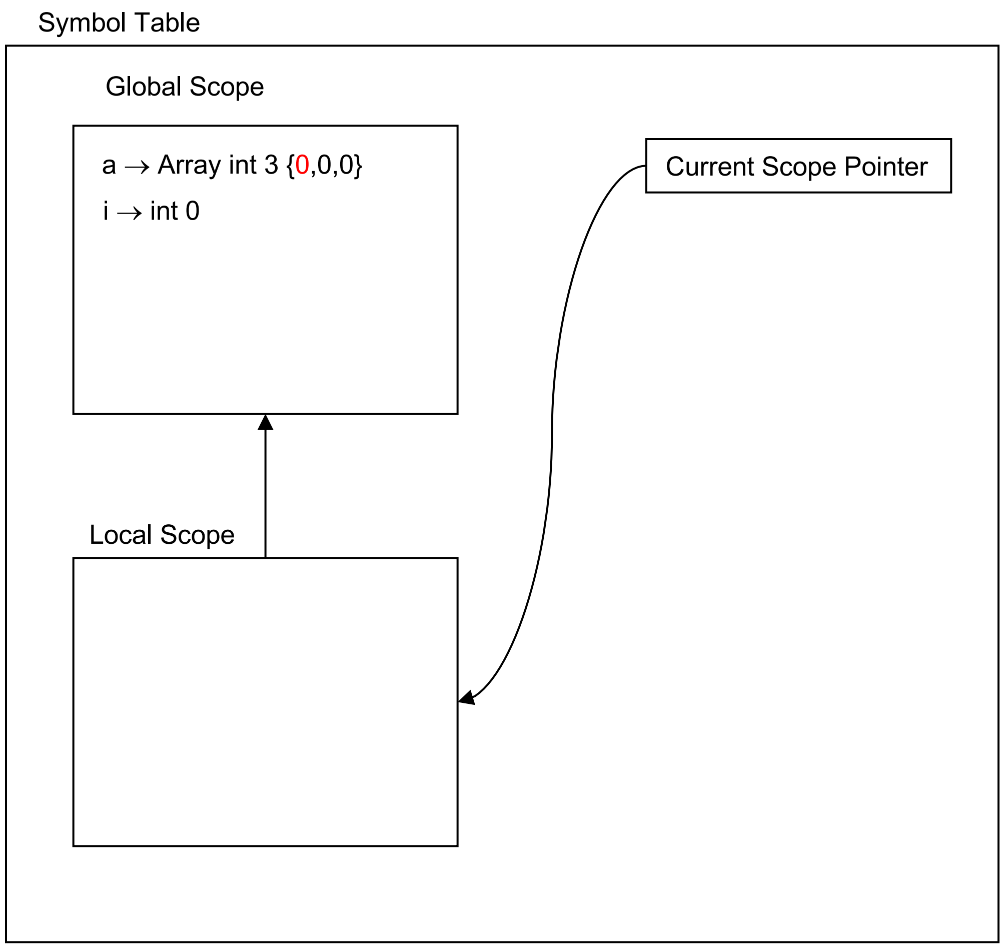
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



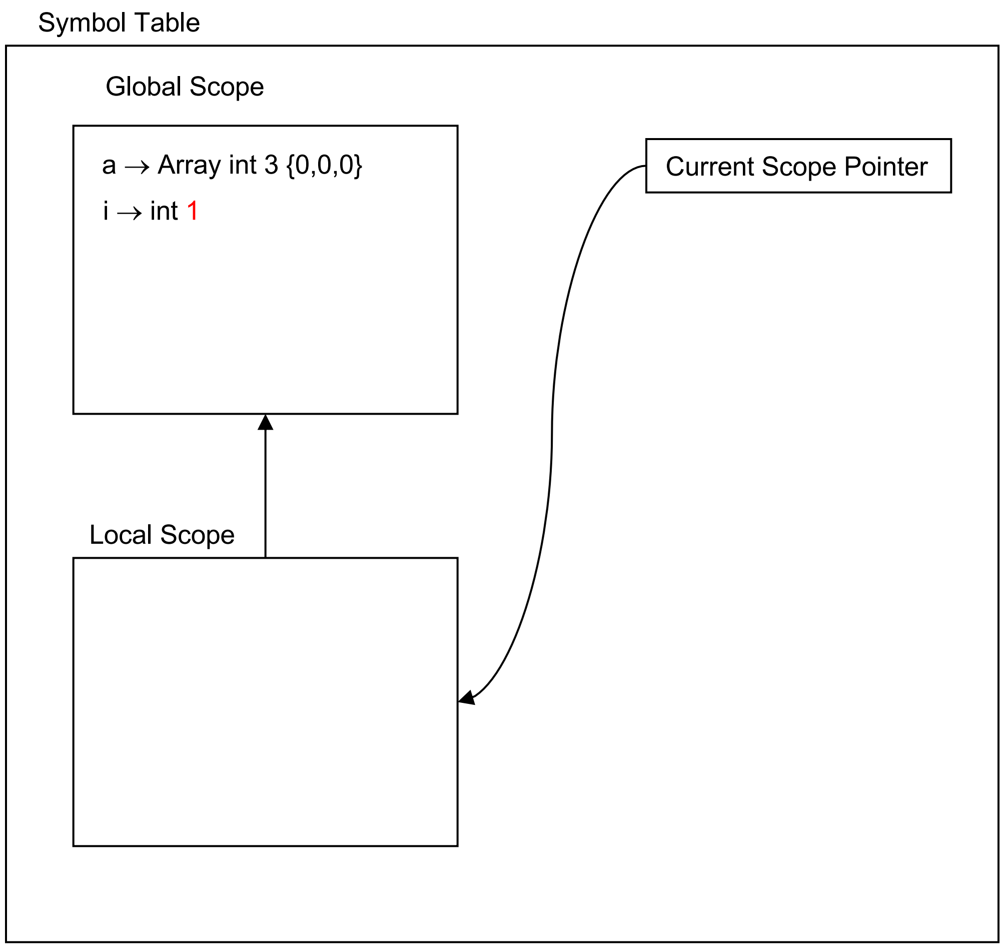
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



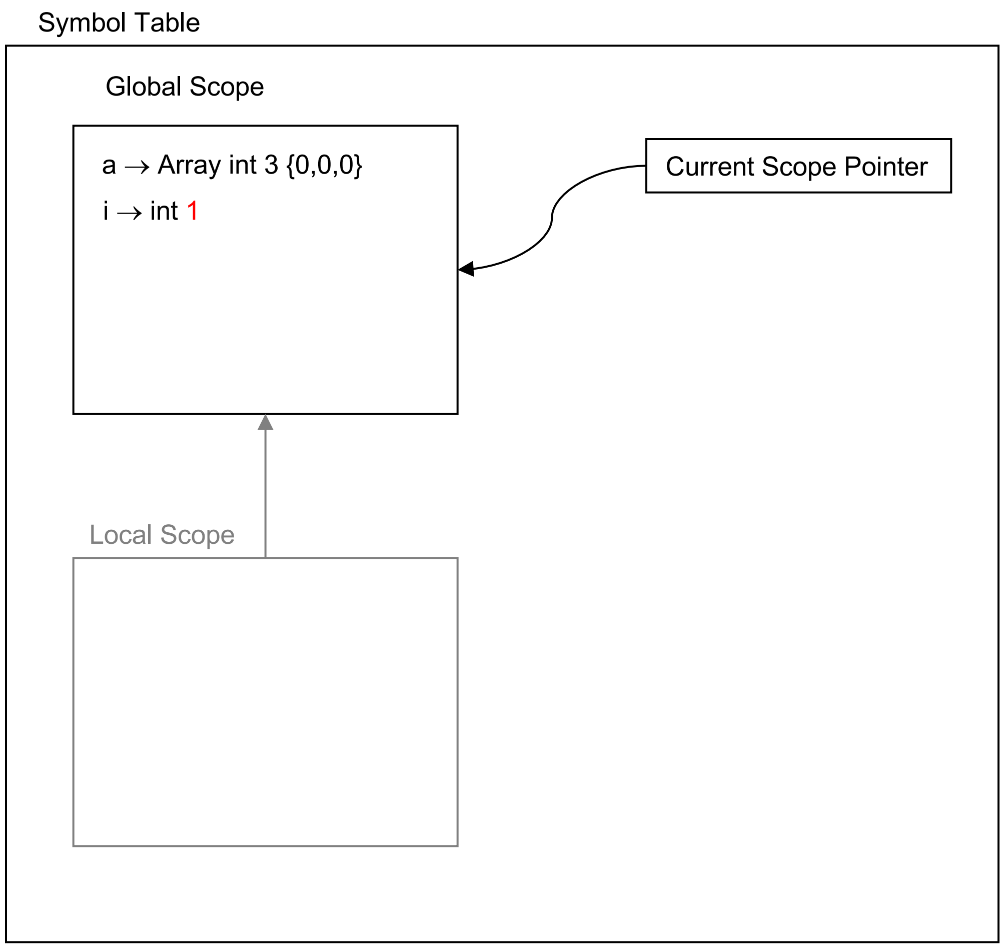
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

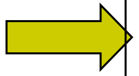
Interpreting Arrays



```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put "the array is: ",a;
```

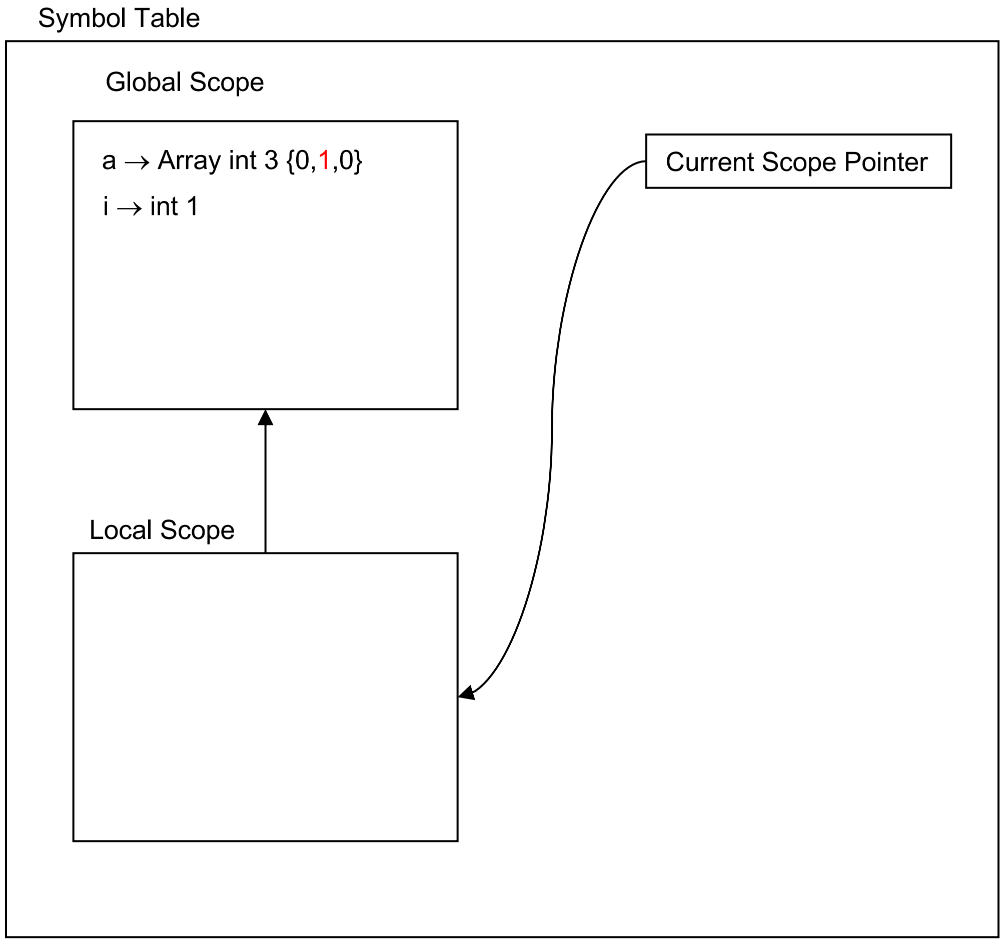
Interpreting Arrays





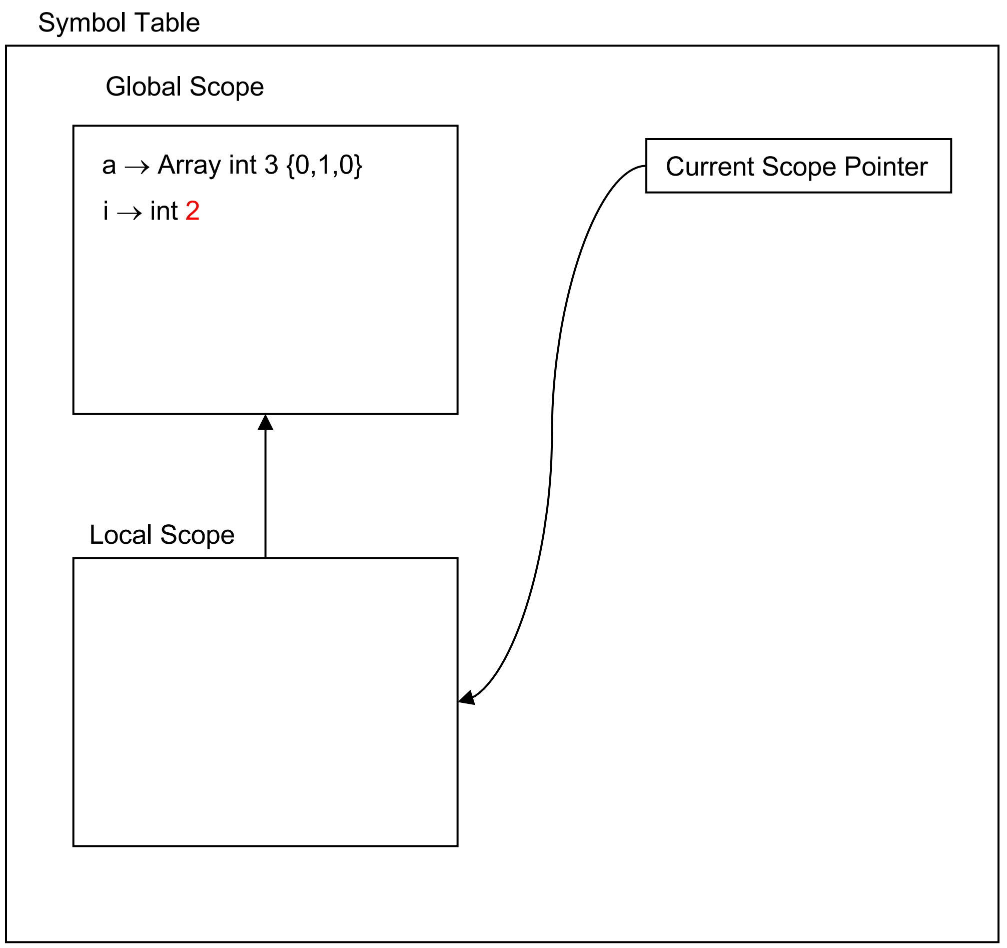
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



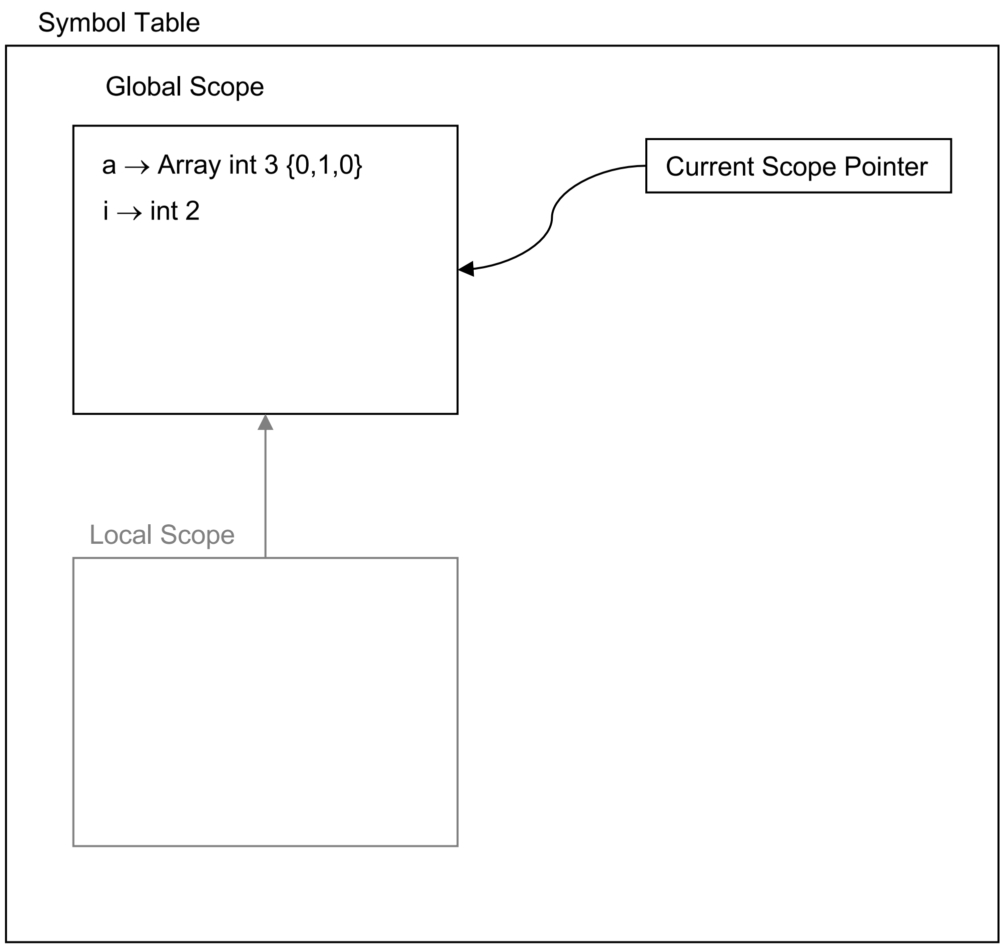
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



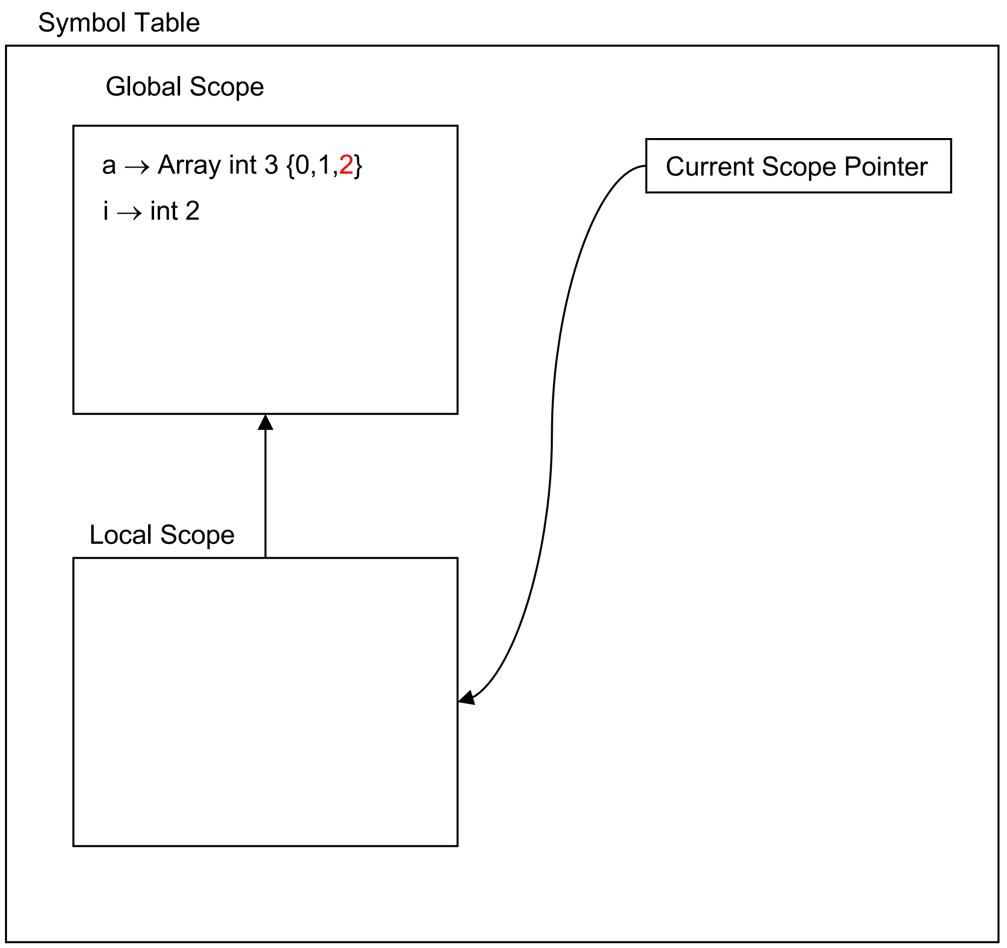
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put "the array is: ",a;
```

Interpreting Arrays



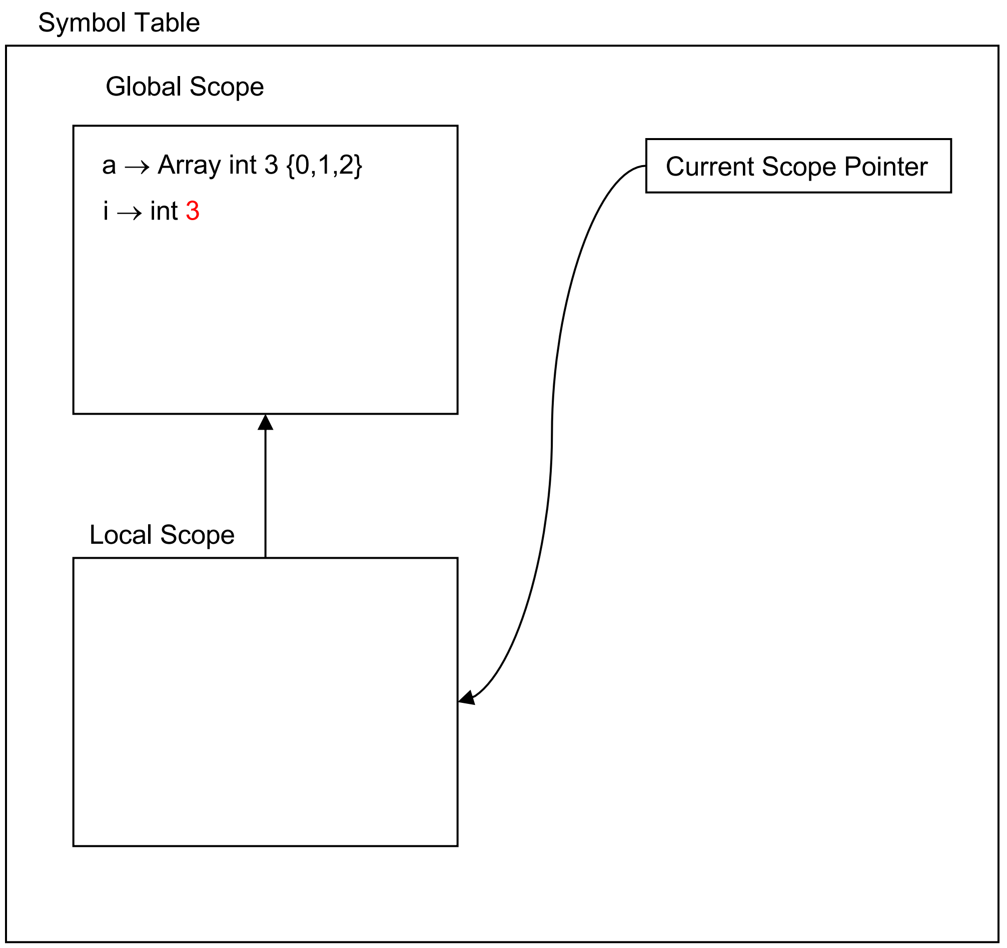
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



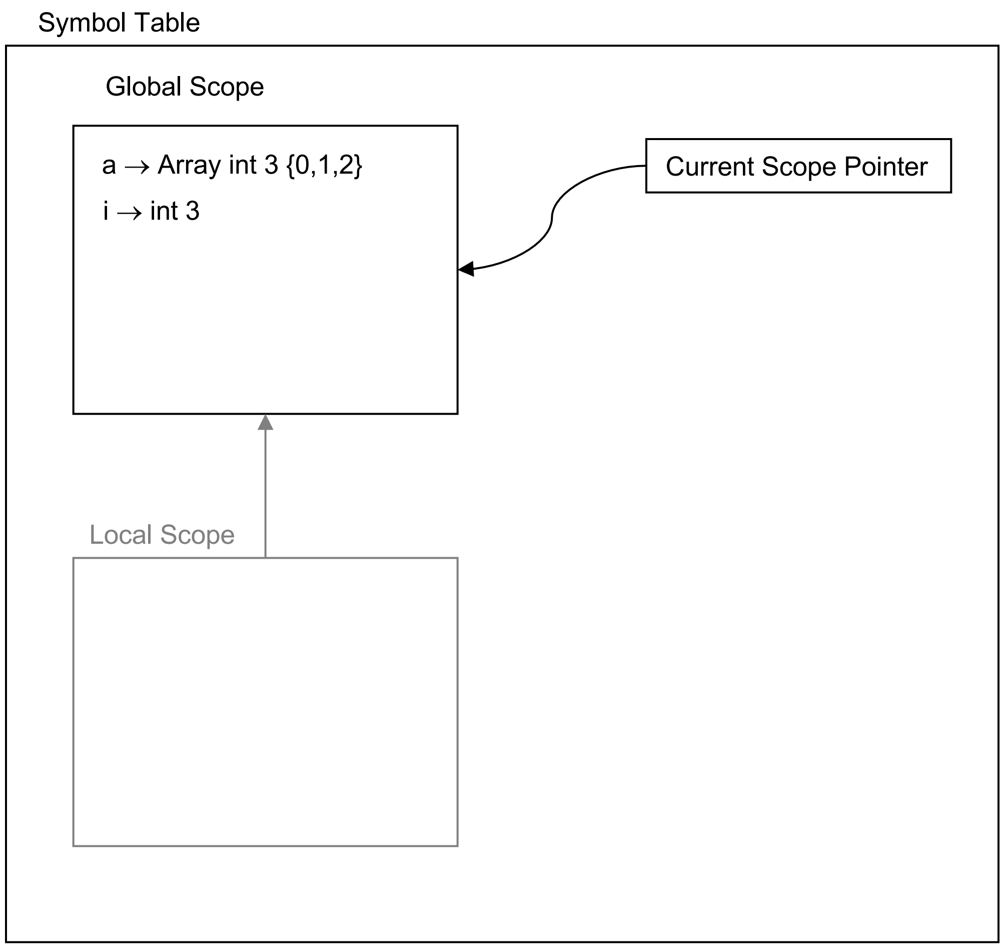
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put "the array is: ",a;
```

Interpreting Arrays



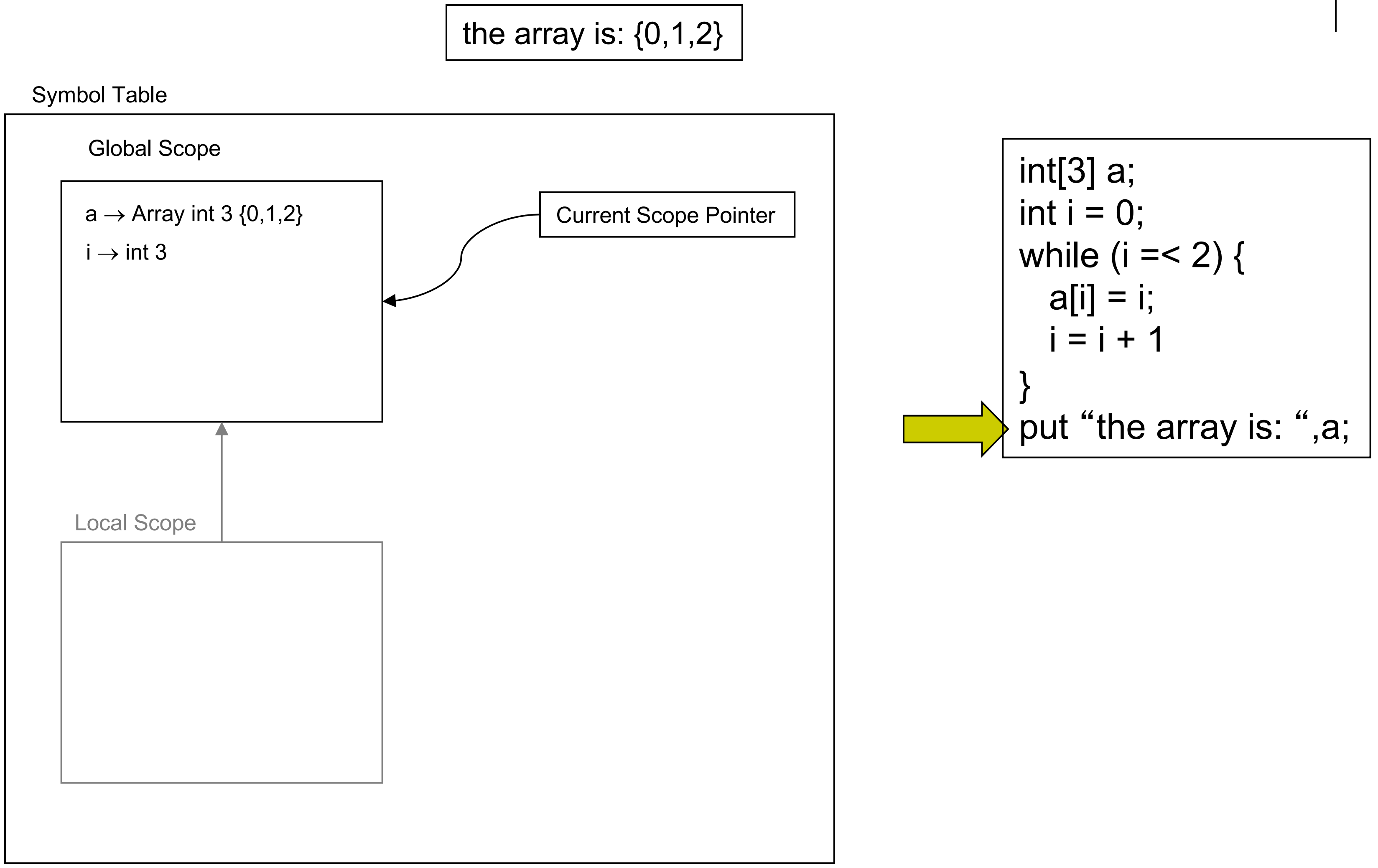
```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put "the array is: ",a;
```

Interpreting Arrays



```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



```
int[3] a;
int i = 0;
while (i =< 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

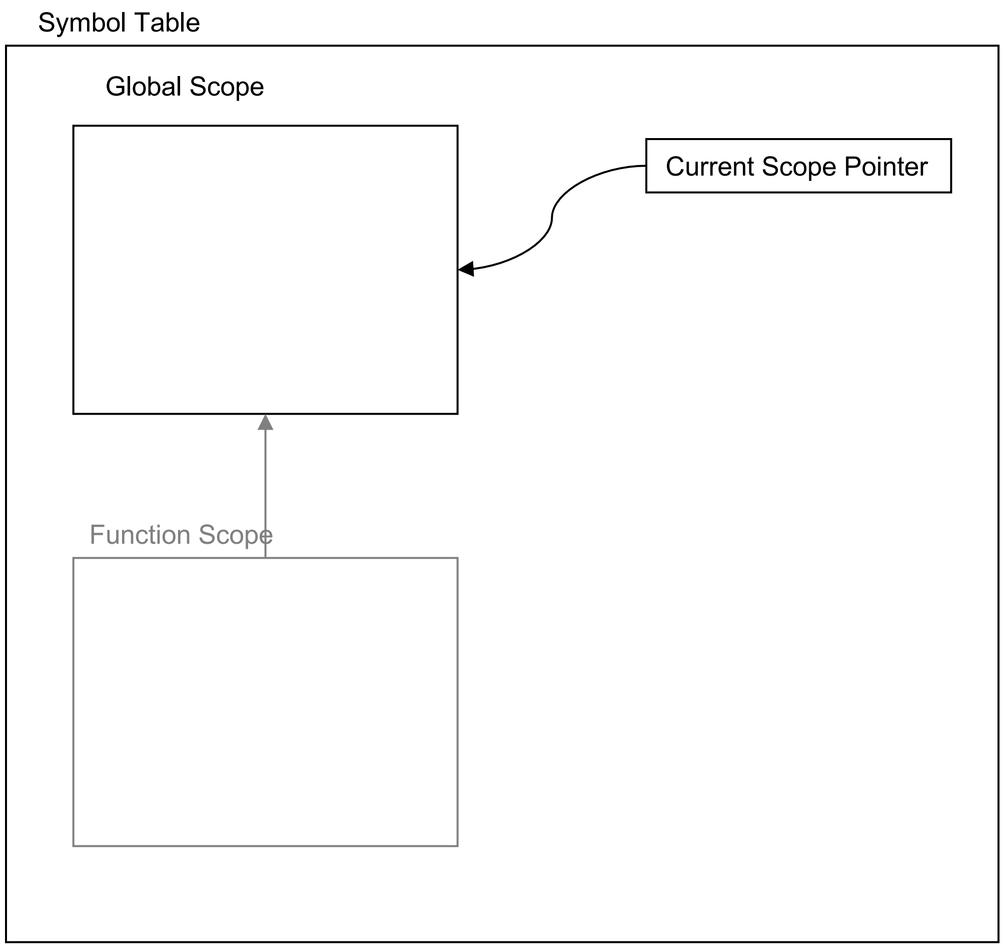
Functions and Arrays

- We pass arrays by-reference to functions
- The types of the formal and actual parameters have to correspond exactly – no type coercion possible.
- We also return arrays from a function by reference.

```
int[3] ident(int[3] a)
{
    return a;
}

int[3] c = {1,2,3};
ident(c)[1] = 0;
put c;
```

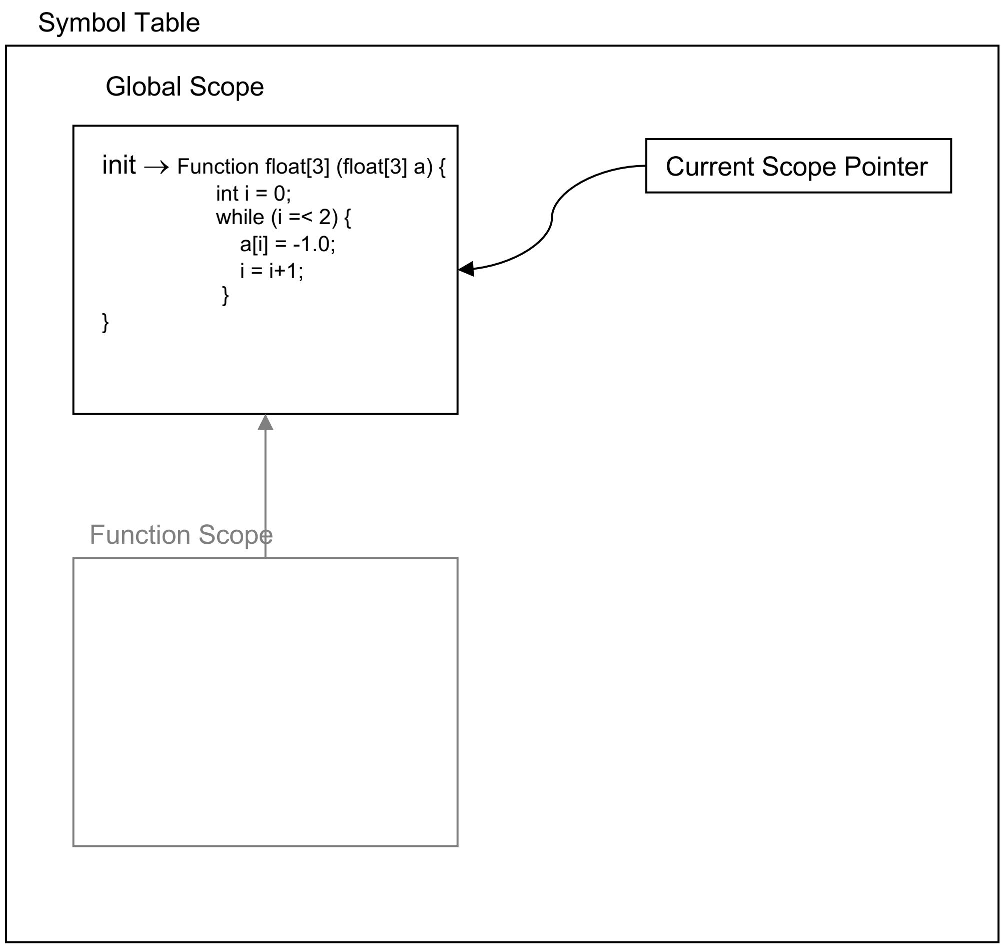
We are modifying c!

Interpreting Arrays



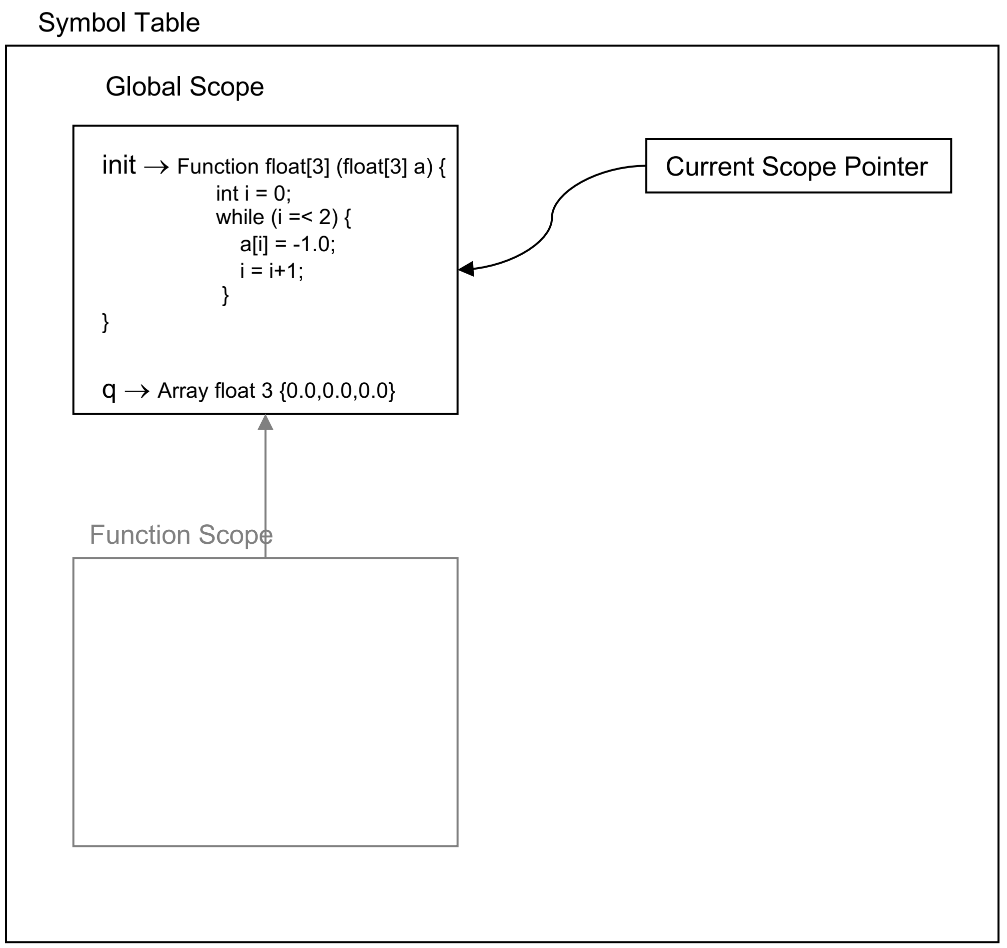
```
float[3] init(float[3] a) {
   int i = 0;
   while (i =< 2) {
      a[i] = -1.0;
      i = i+1;
   }
}

float[3] q;
init(q);
```

Interpreting Arrays



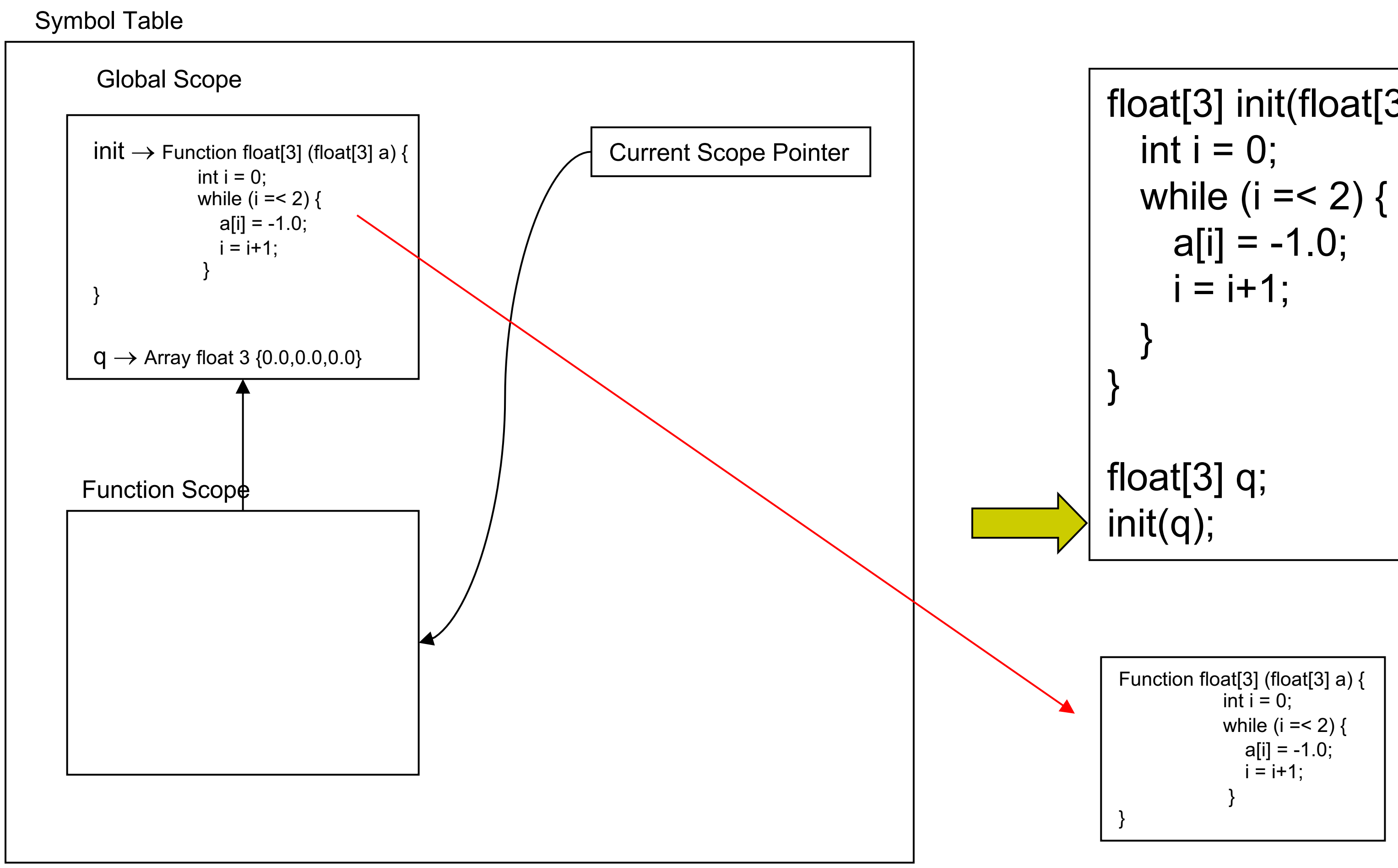
```
float[3] init(float[3] a) {
  int i = 0;
  while (i =< 2) {
    a[i] = -1.0;
    i = i+1;
  }
}

float[3] q;
init(q);
```

Interpreting Arrays



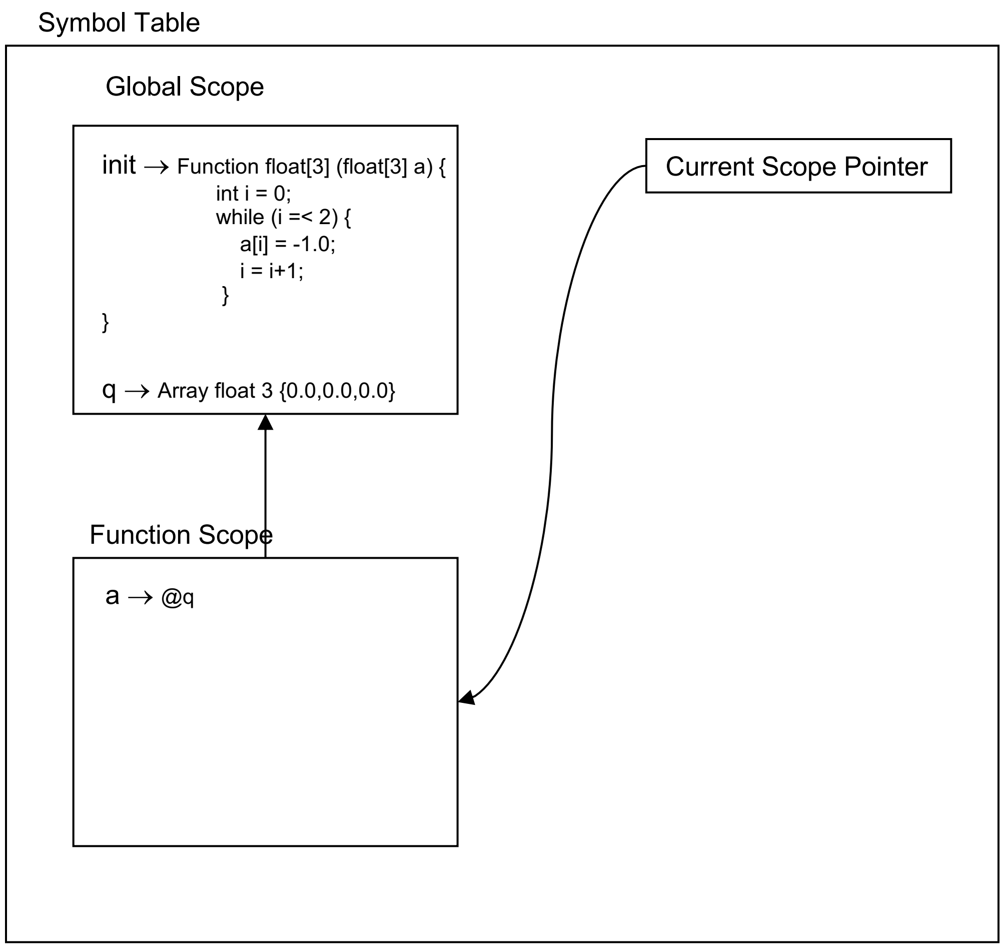
```
float[3] init(float[3] a) {
   int i = 0;
   while (i =< 2) {
      a[i] = -1.0;
      i = i+1;
   }
}

float[3] q;
init(q);
```

Interpreting Arrays

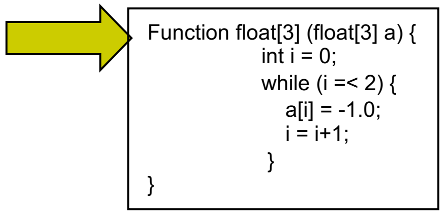


```
float[3] init(float[3] a) {
   int i = 0;
   while (i =< 2) {
      a[i] = -1.0;
      i = i+1;
   }
}

float[3] q;
init(q);
```

Interpreting Arrays

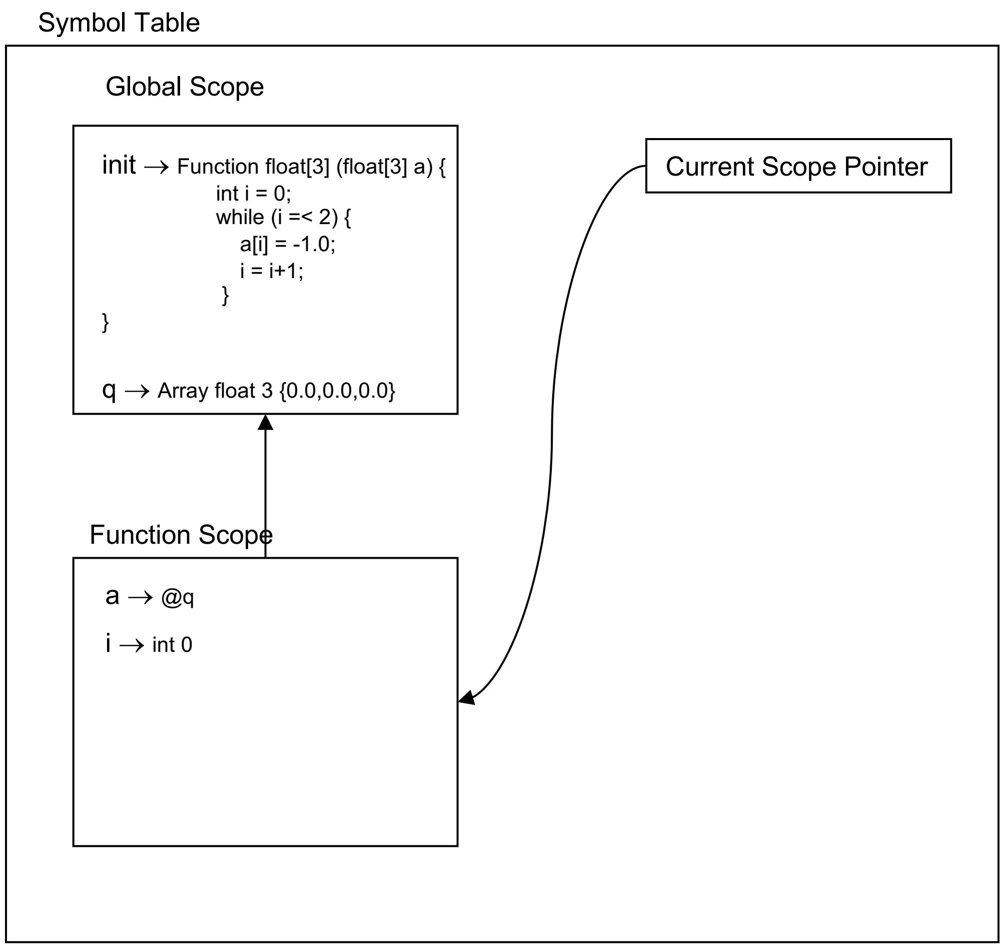


```
float[3] init(float[3] a) {
   int i = 0;
   while (i =< 2) {
      a[i] = -1.0;
      i = i+1;
   }
}

float[3] q;
init(q);
```



```
Function float[3] (float[3] a) {
            int i = 0;
            while (i =< 2) {
               a[i] = -1.0;
               i = i+1;
            }
}
```

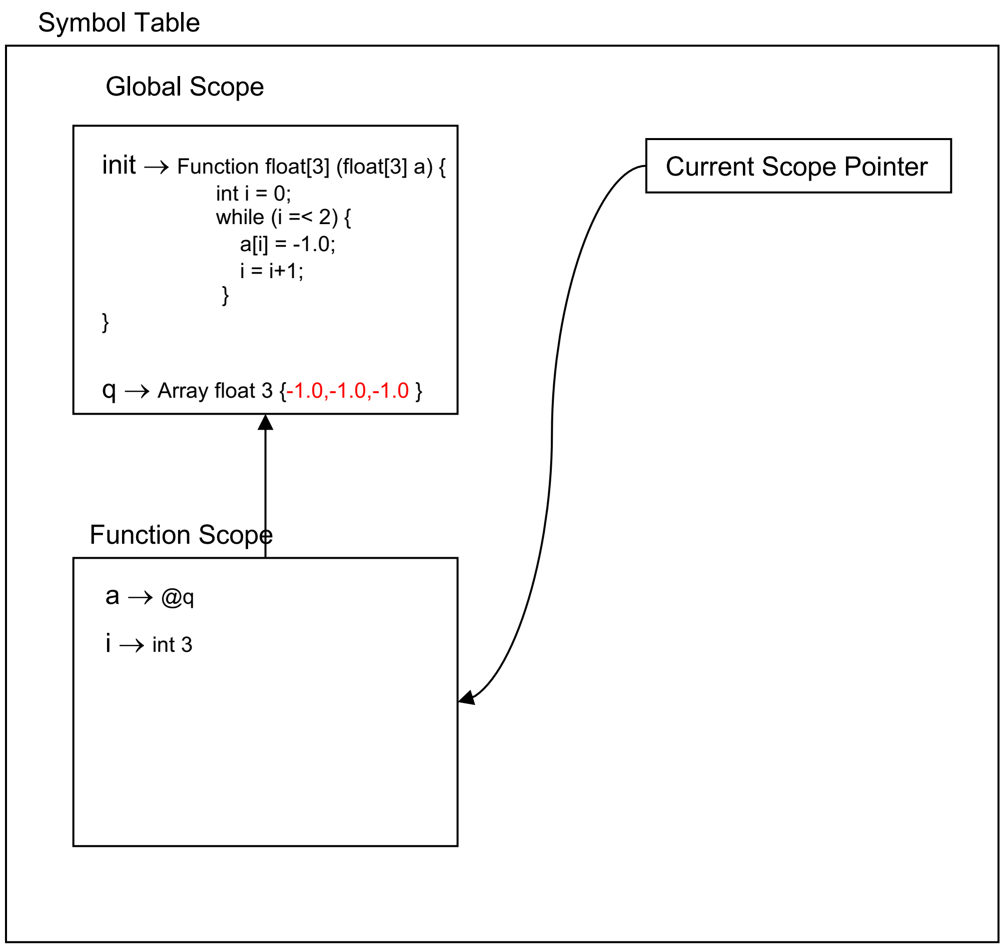
Interpreting Arrays



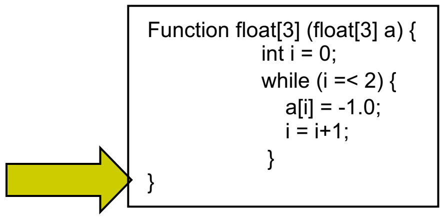
```
float[3] init(float[3] a) {
   int i = 0;
   while (i =< 2) {
      a[i] = -1.0;
      i = i+1;
   }
}

float[3] q;
init(q);
```

```
Function float[3] (float[3] a) {
              int i = 0;
              while (i =< 2) {
                 a[i] = -1.0;
                 i = i+1;
              }
}
```

Interpreting Arrays

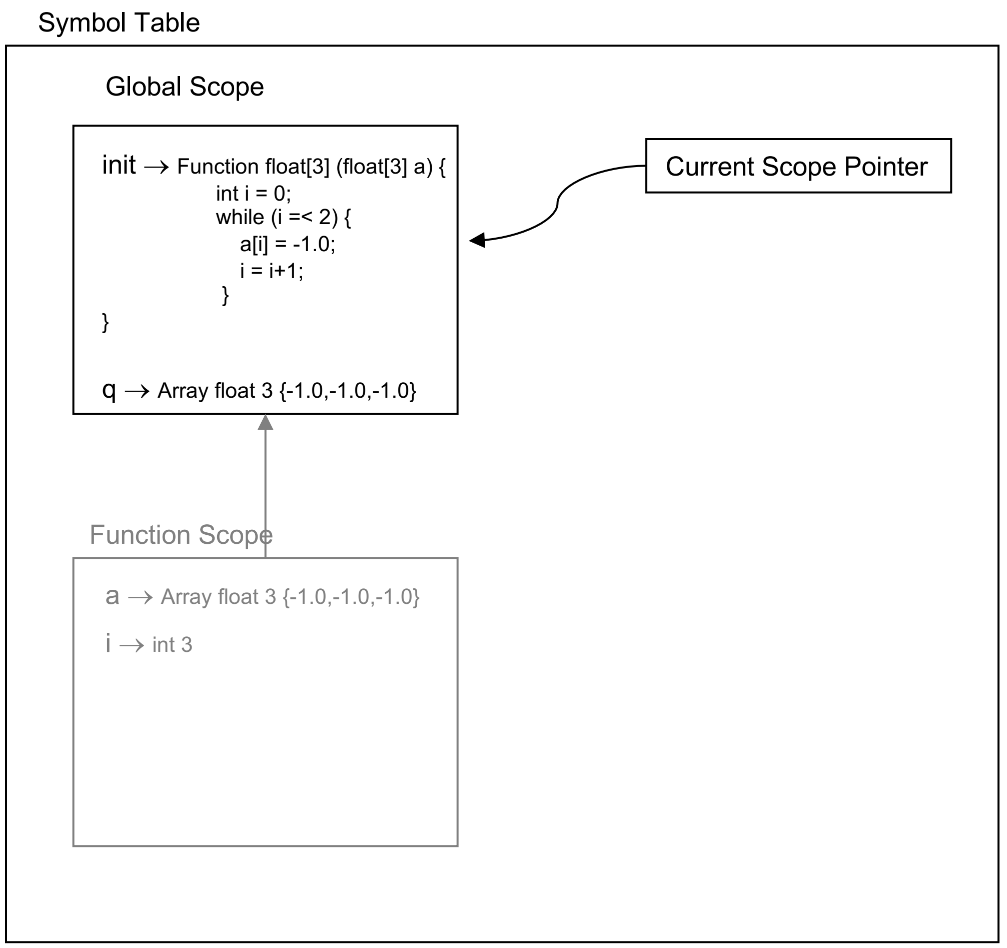


```
float[3] init(float[3] a) {
  int i = 0;
  while (i =< 2) {
    a[i] = -1.0;
    i = i+1;
  }
}

float[3] q;
init(q);
```



```
Function float[3] (float[3] a) {
             int i = 0;
             while (i =< 2) {
                a[i] = -1.0;
                i = i+1;
             }
}
```

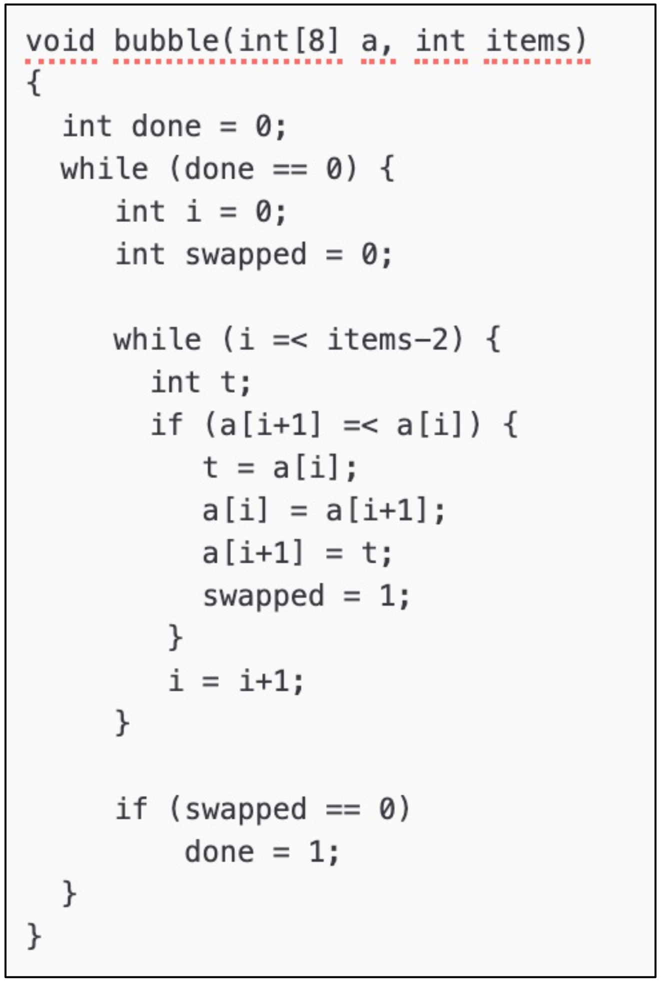
Interpreting Arrays



```
float[3] init(float[3] a) {
   int i = 0;
   while (i =< 2) {
      a[i] = -1.0;
      i = i+1;
   }
}

float[3] q;
init(q);
```

Computing with Arrays

- The Bubble Sort



```
void bubble(int[8] a, int items)
{
  int done = 0;
  while (done == 0) {
     int i = 0;
     int swapped = 0;

     while (i =< items-2) {
       int t;
       if (a[i+1] =< a[i]) {
          t = a[i];
          a[i] = a[i+1];
          a[i+1] = t;
          swapped = 1;
        }
        i = i+1;
     }

     if (swapped == 0)
         done = 1;
  }
}
```