Structured Data Types

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

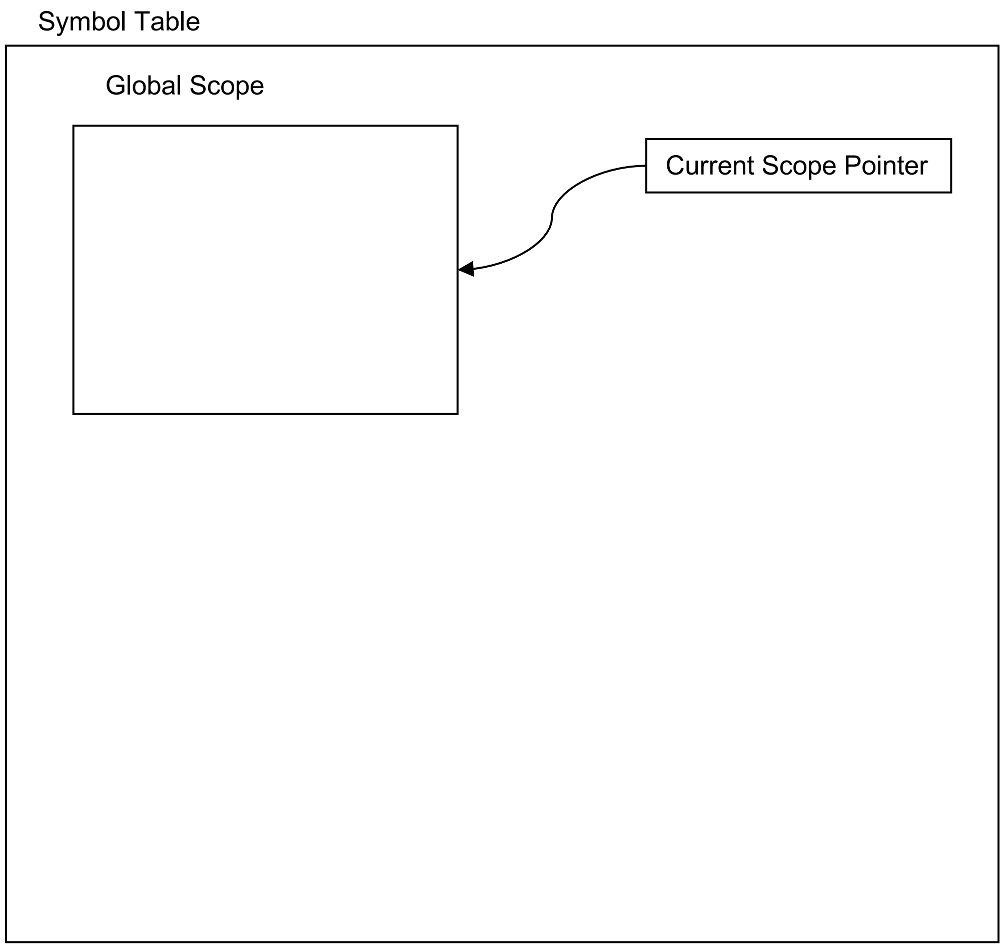
Arrays

- Arrays are data structures that look like lists where every element in the list is of the same data type.
- Arrays consist of an *array variable* and an *array index*
- 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

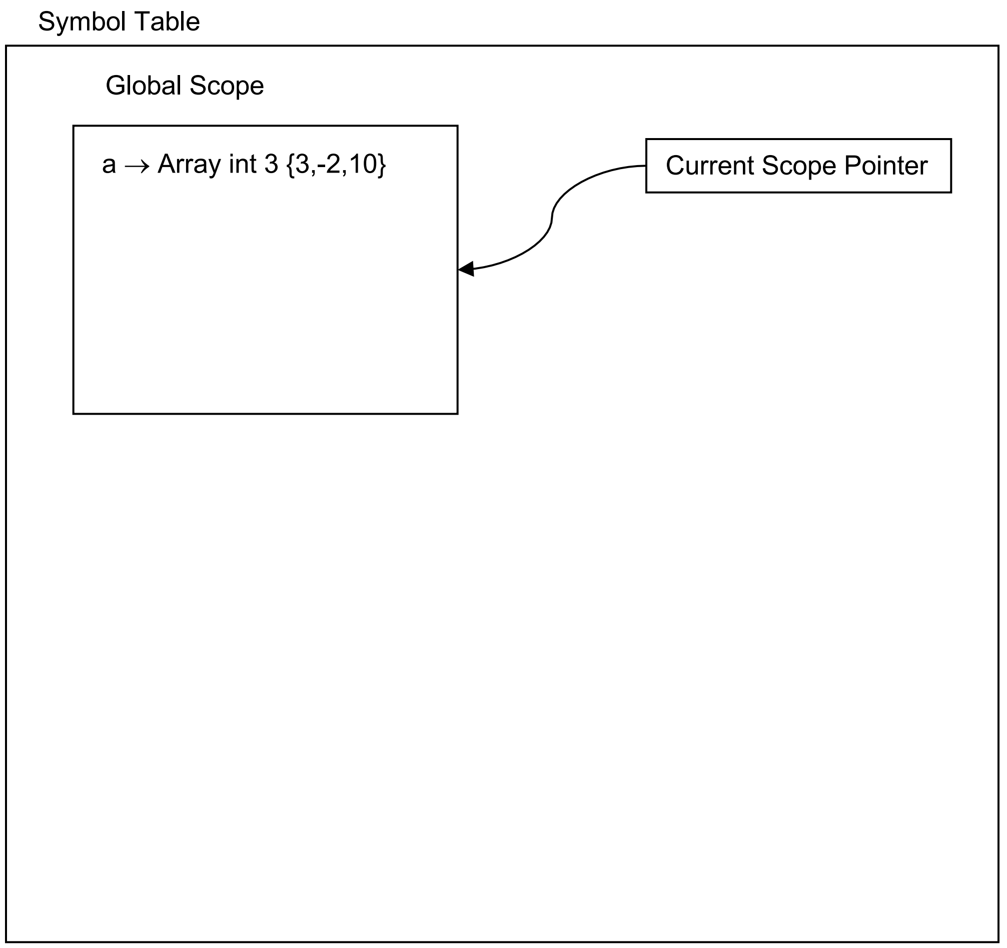
- Initializers
  - int[3] a = { 3,-2,10 };
- Arrays can be viewed as *array values*
  - int[3] a = { 3,-2,10 };
  - int[3] b = a;
- The size of the array and the type of the elements matters
  - int[3] a = { 3,-2,10};
  - float[3] b = a; ✗
  - or
  - int[4] b = a; ✗

Interpreting Arrays



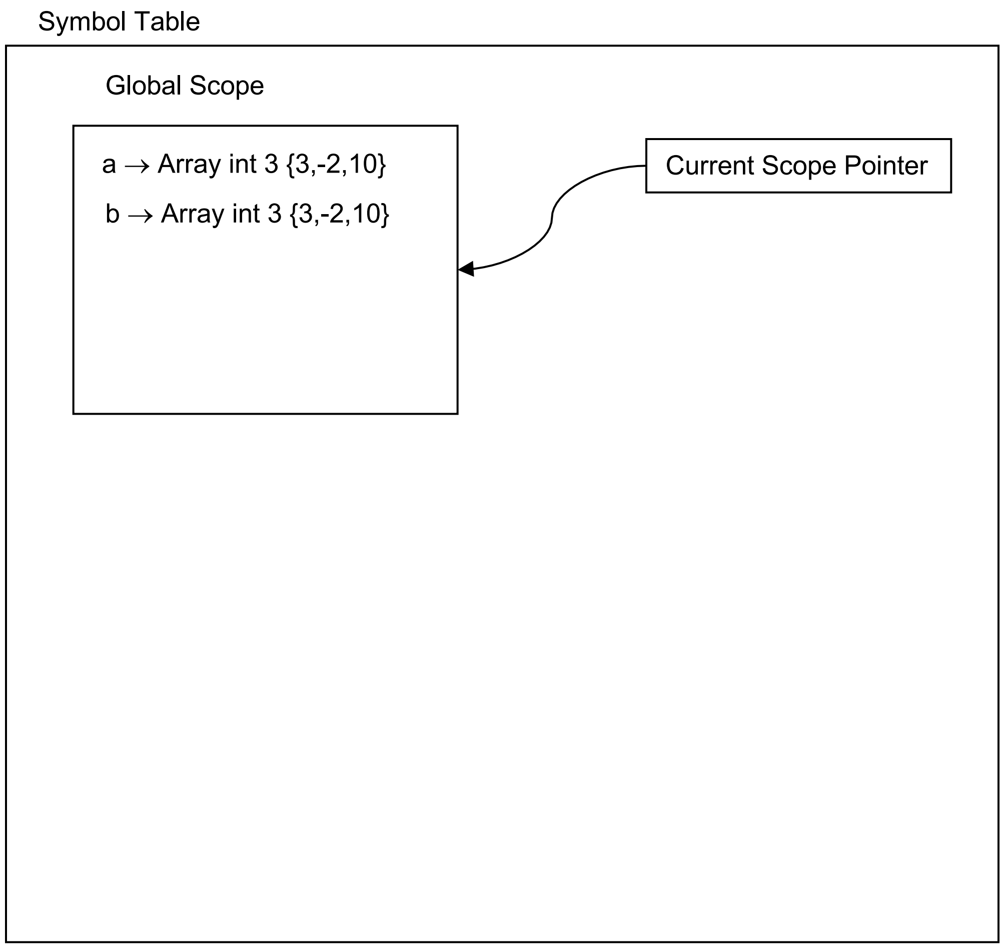
```
int[3] a = { 3,-2,10 };
int[3] b = a;
```

Interpreting Arrays



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

Interpreting Arrays



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

Computing with Arrays

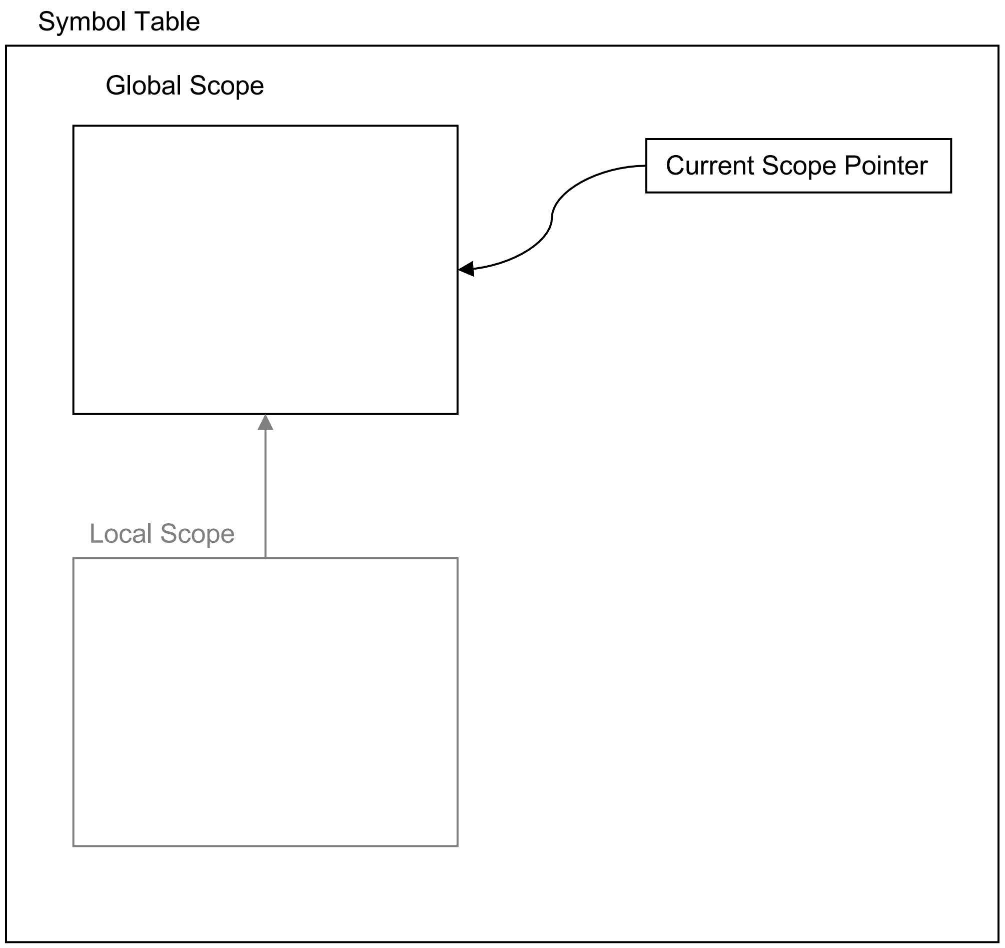
- Just as in the case of scalar variables, array variables can appear in two types of contexts:
  - Expressions: here we read the contents of the array location indexed, e.g., x = a[2].
  - Assignment statements: here we access the index array location and update its contents, e.g., a[2] = x

Computing with Arrays

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

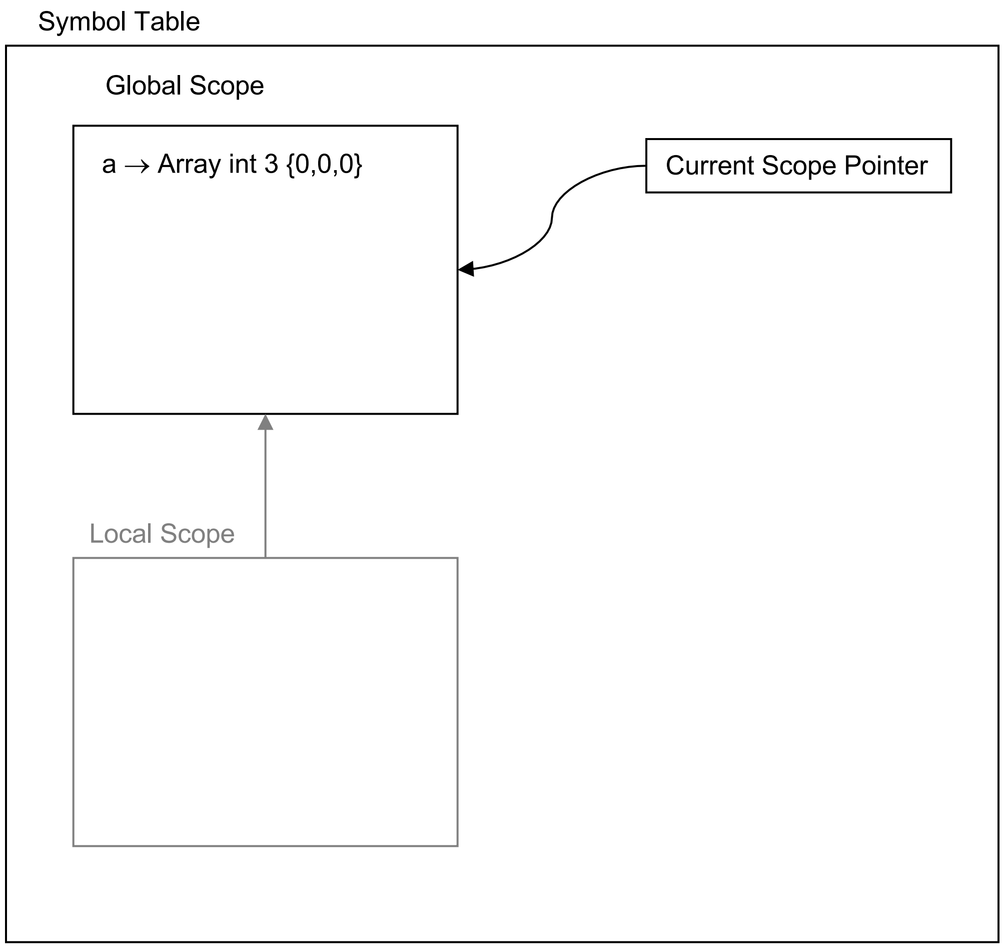
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “, a;
```

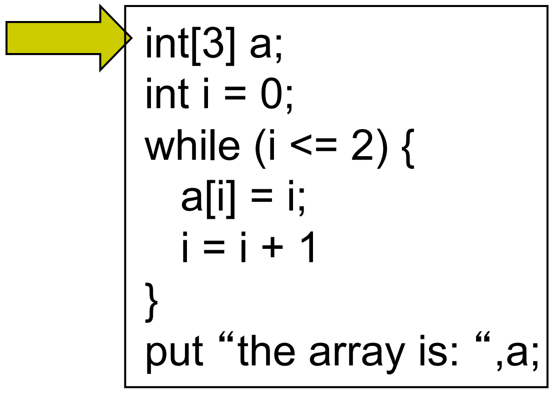
Interpreting Arrays



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

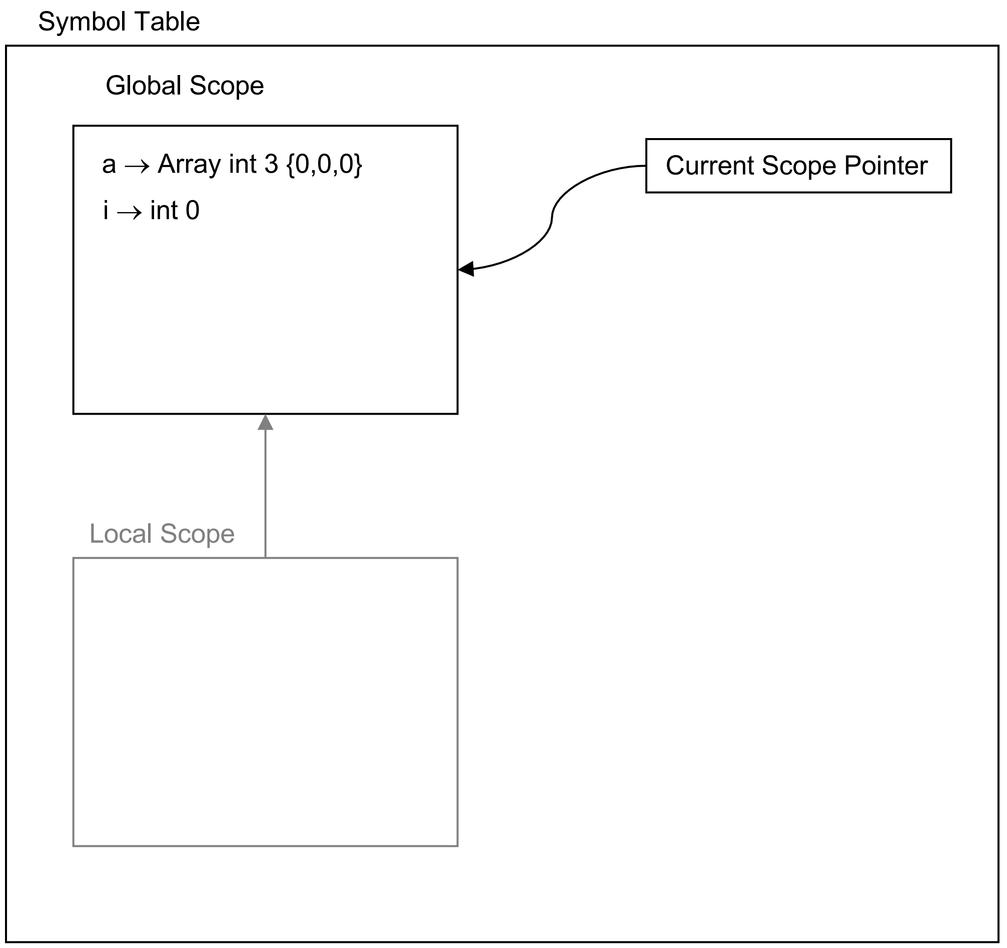
Interpreting Arrays





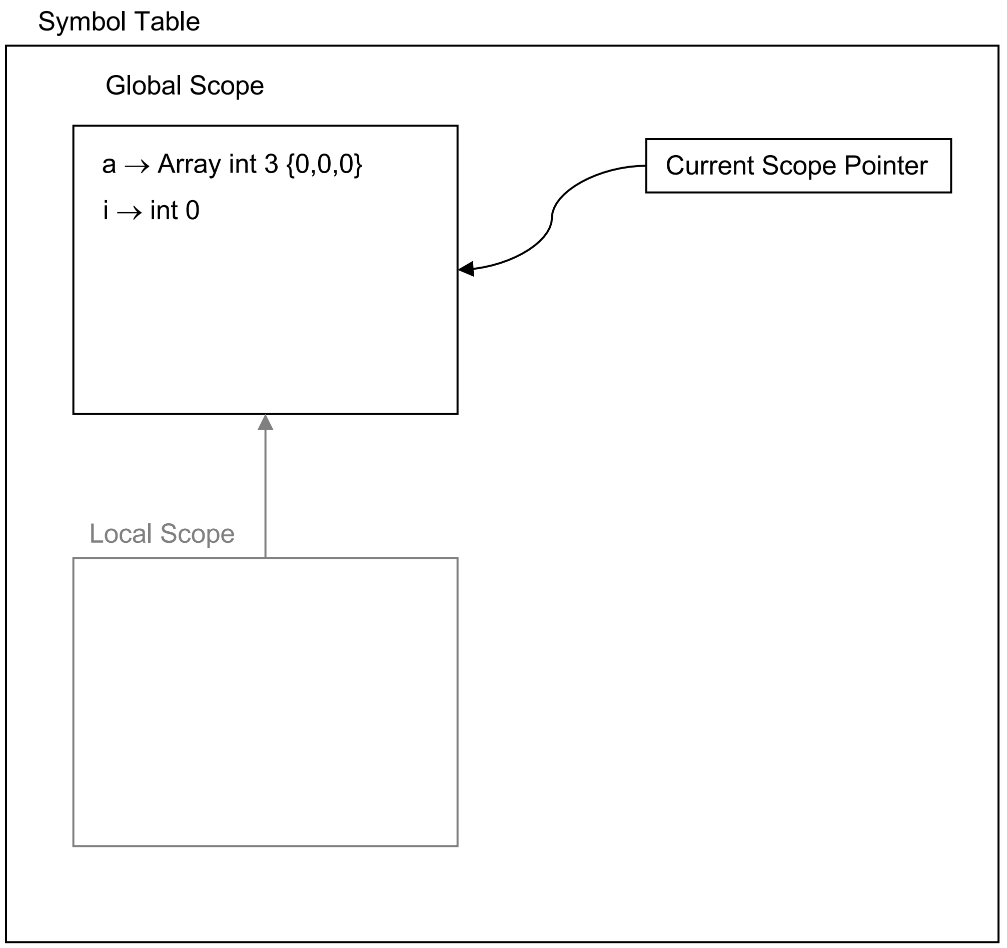
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



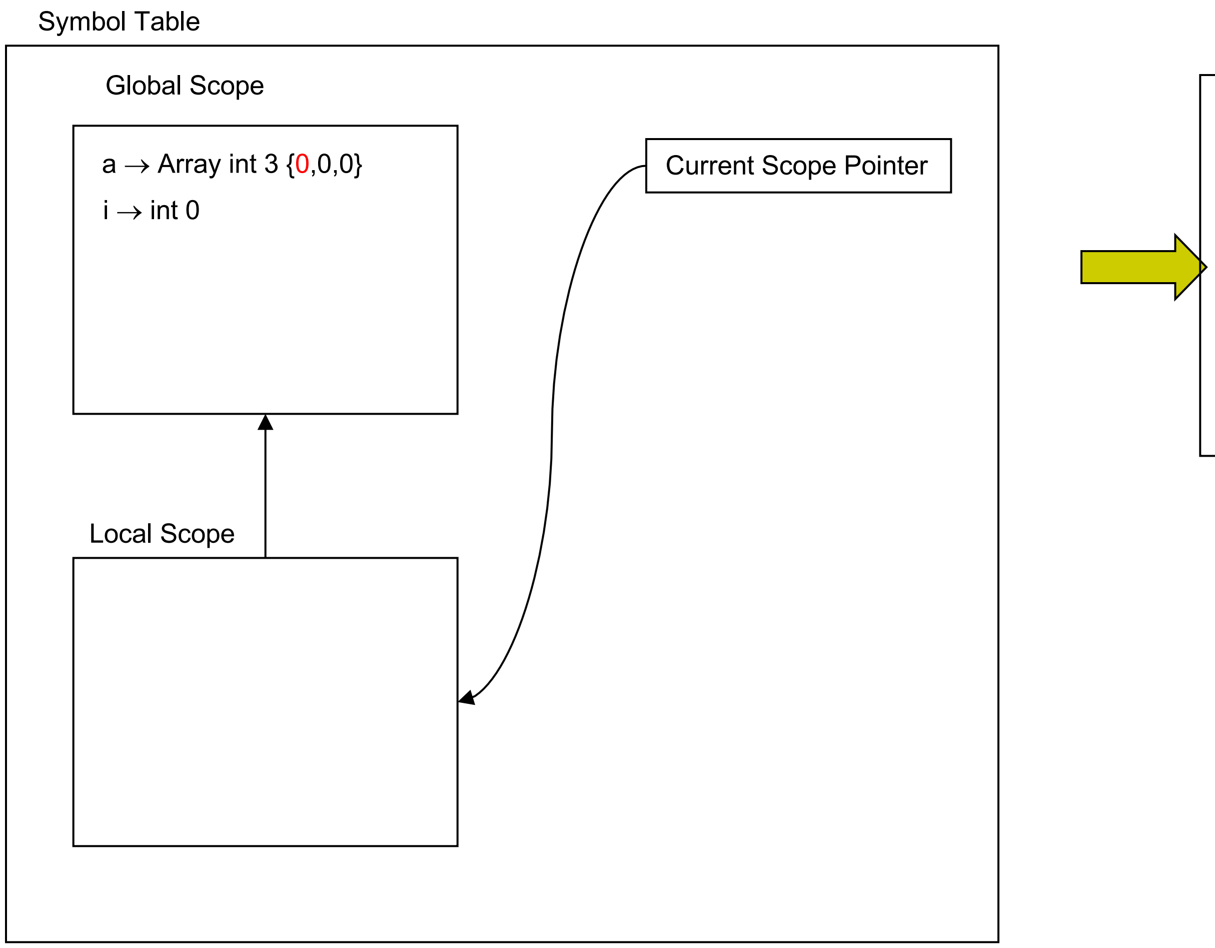
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



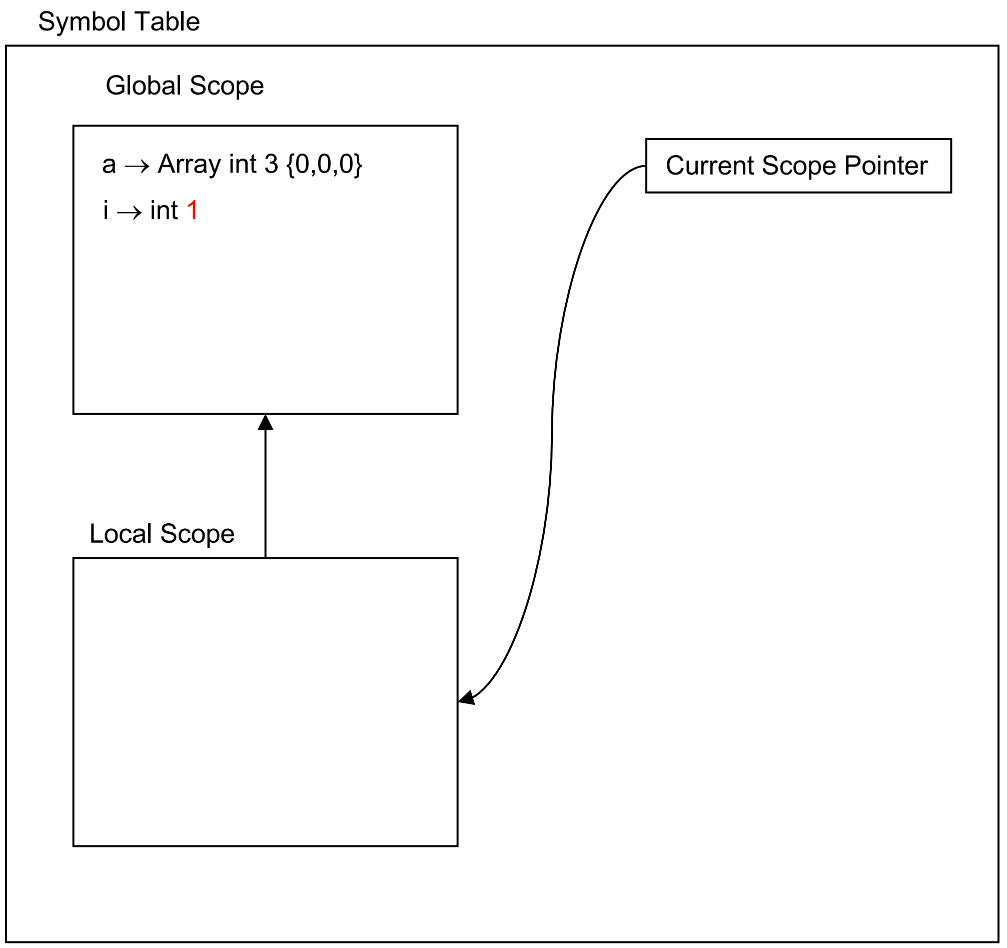
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put "the array is: ",a;
```

Interpreting Arrays



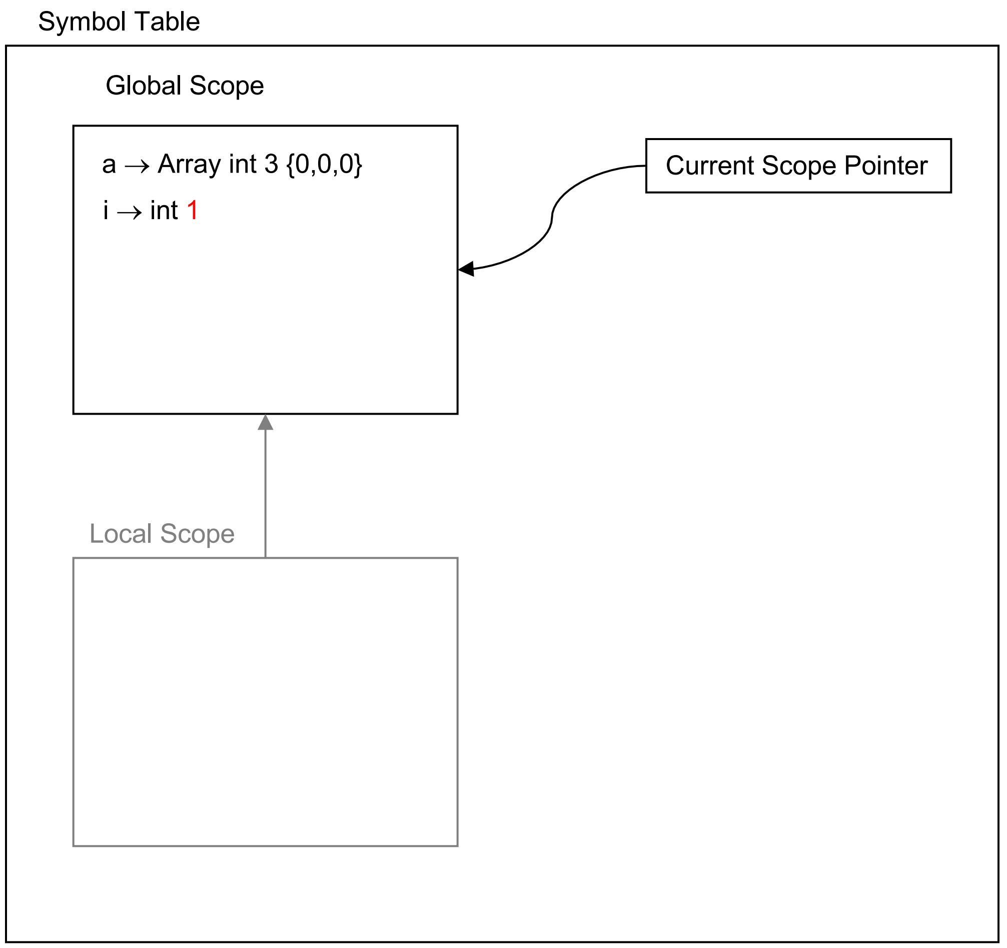
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



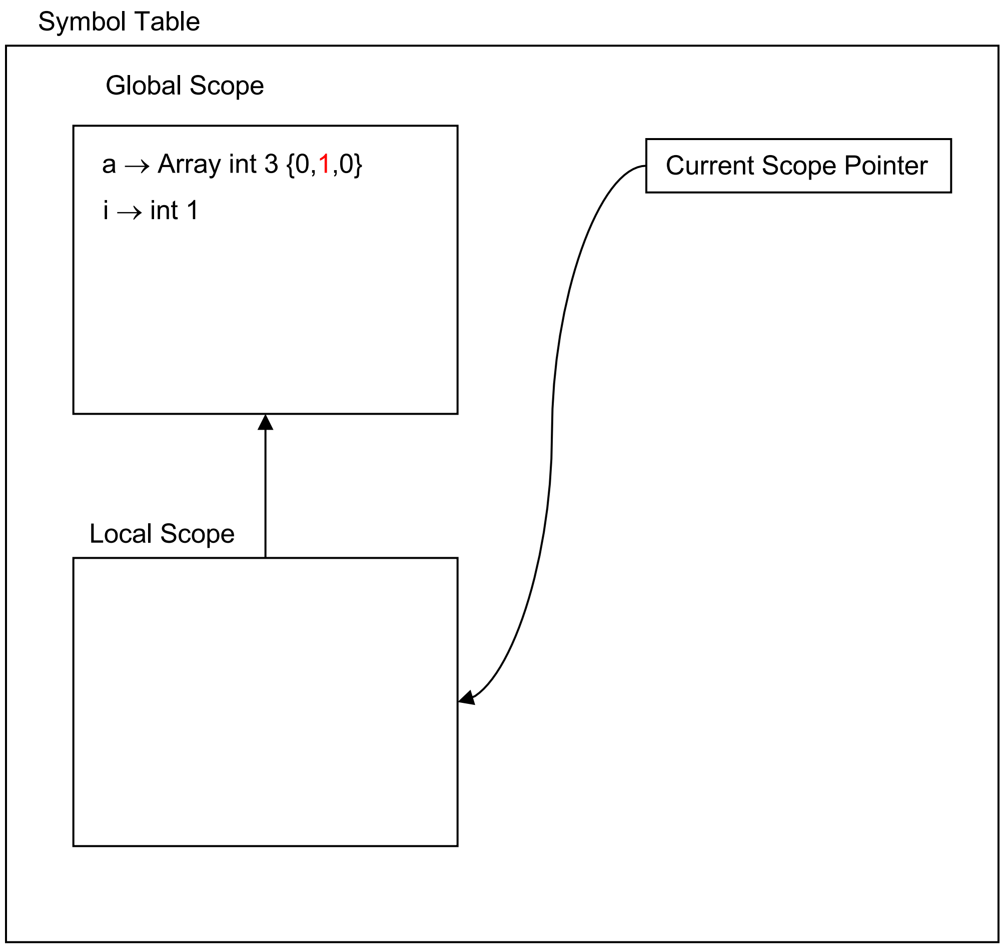
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



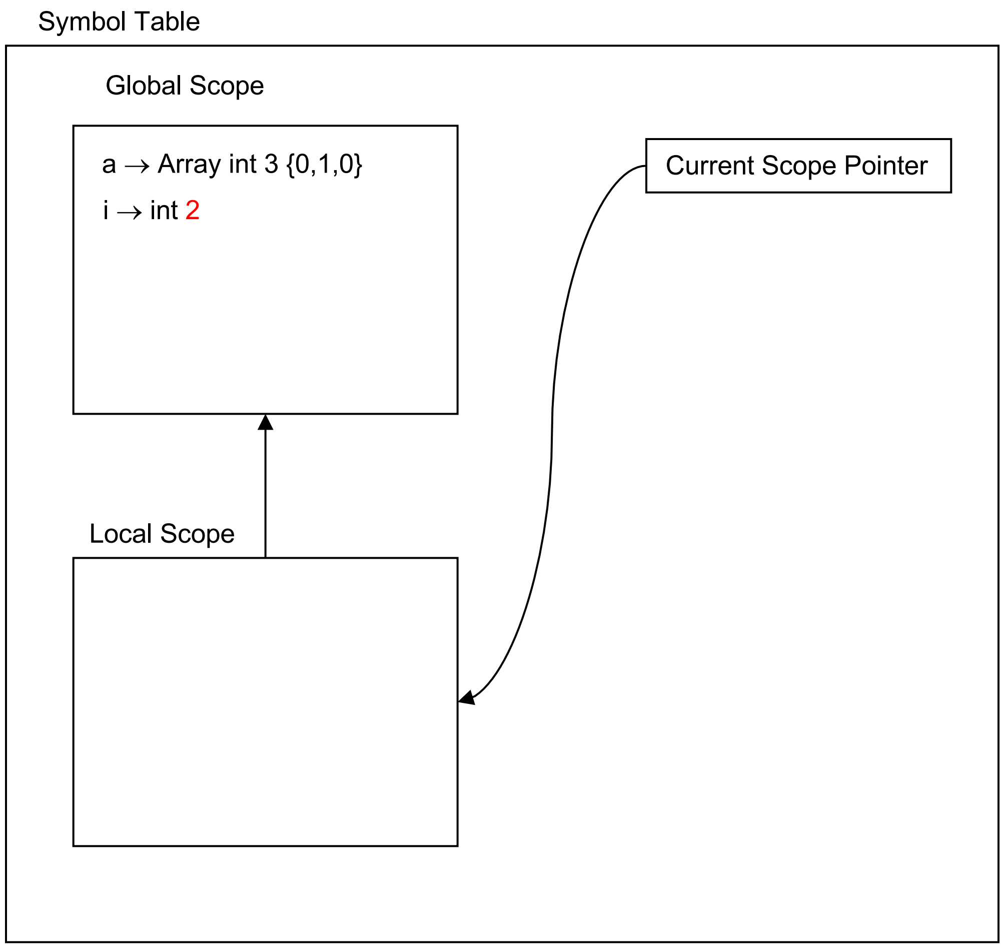
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



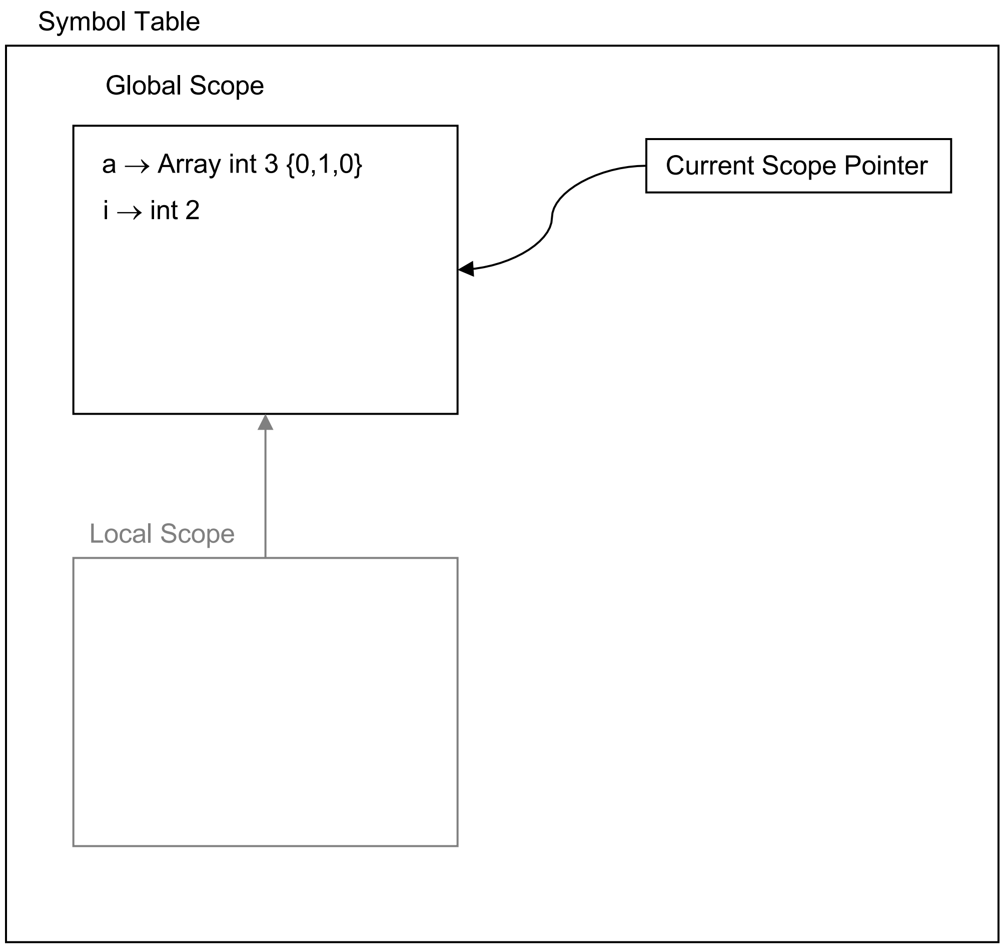
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



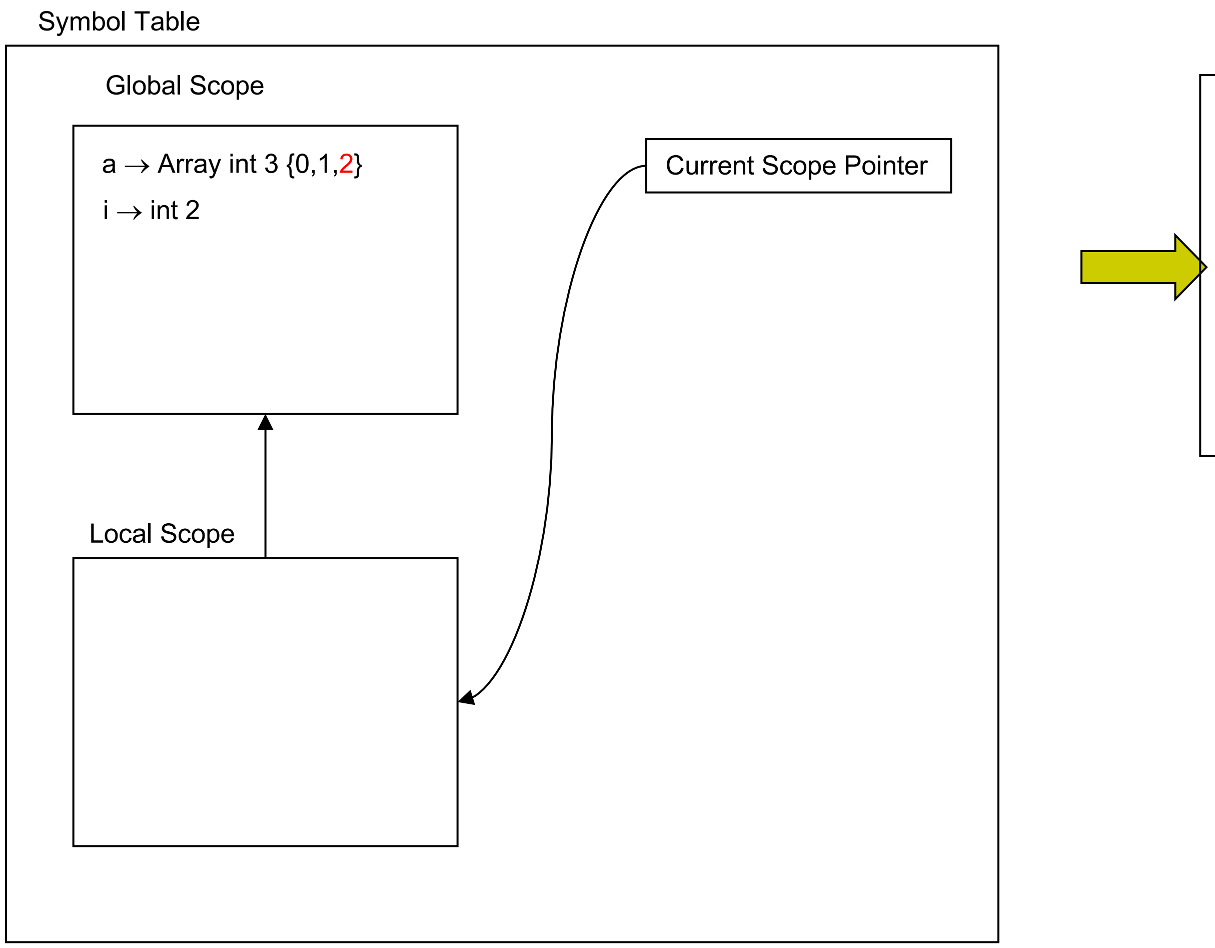
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



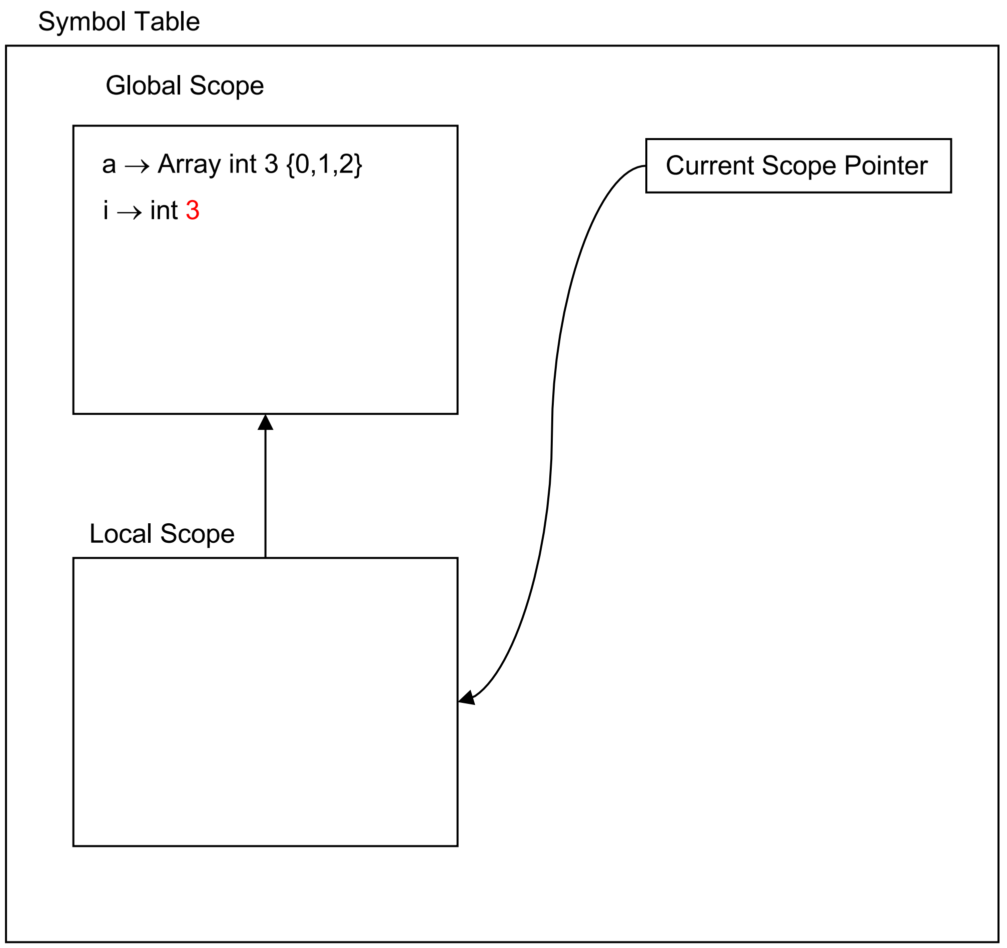
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



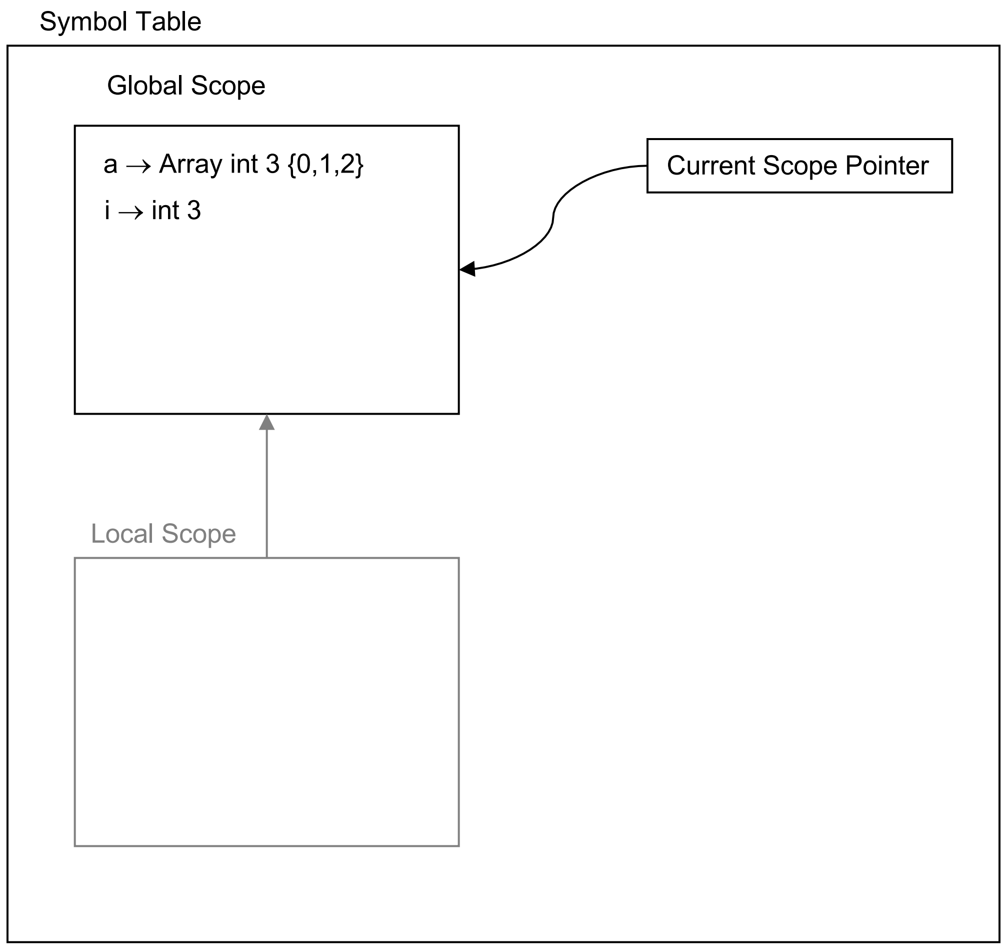
```
int[3] a;
int i = 0;
while (i <= 2) {
   a[i] = i;
   i = i + 1
}
put “the array is: “,a;
```

Interpreting Arrays



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

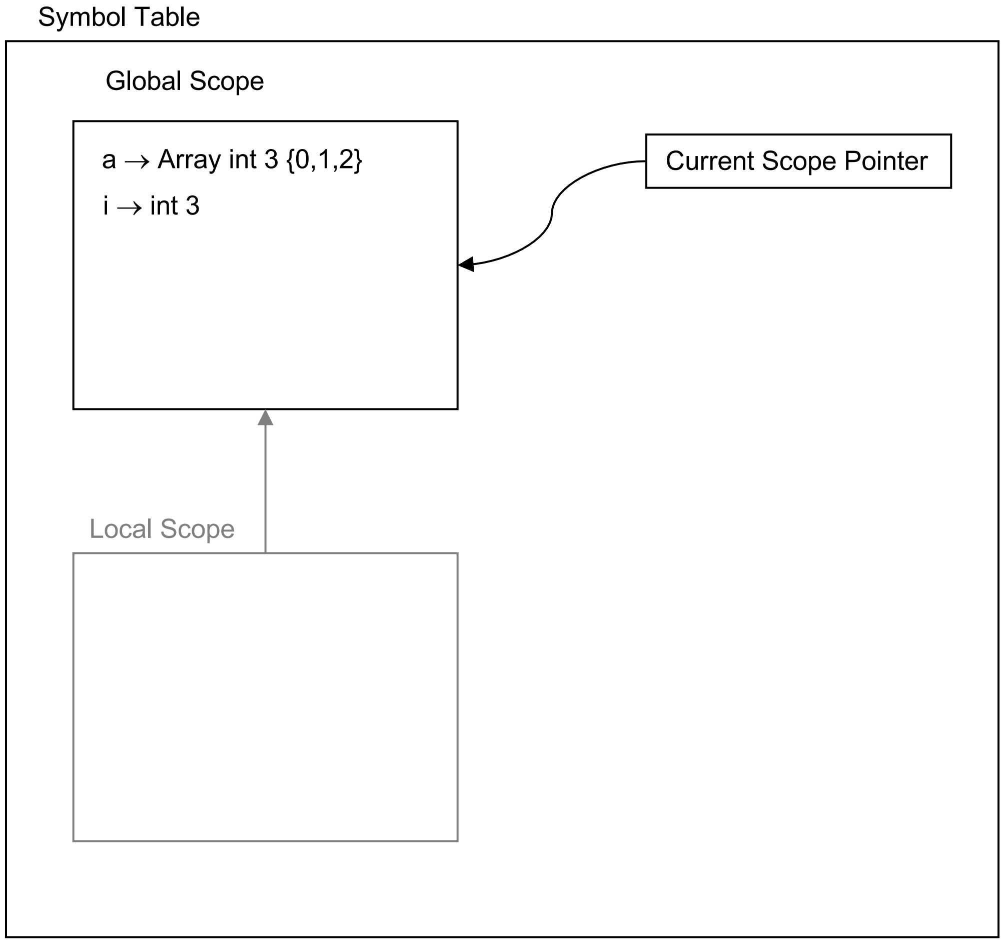
Interpreting Arrays



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

Interpreting Arrays

the array is: {0,1,2}



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

Computing with Arrays

- The Bubble Sort

6 5 3 1 8 7 2 4

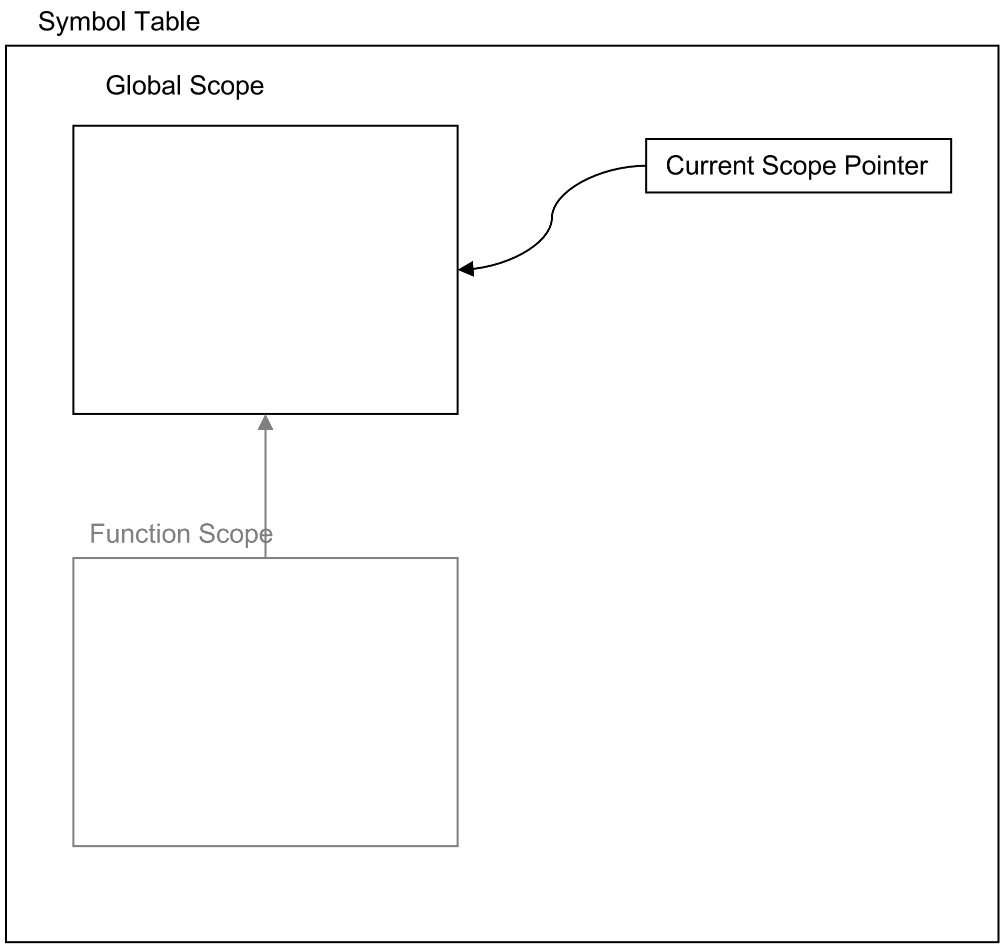
```
int[8] a = {6,5,3,1,8,7,2,4};
int done = 0;

while (done == 0) {
   int i = 0;
   int swapped = 0;

   while (i <= 6) {
      if (a[i+1] <= a[i]) {
         int t = a[i];
         a[i] = a[i+1];
         a[i+1] = t;
         swapped = 1;
      }
      i = i+1;
   }

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

put a;
```

Functions and Arrays

- Since we interpret arrays as values we can easily pass them to functions as long as the formal parameters are declared properly
- Returning an array from a function is similar.

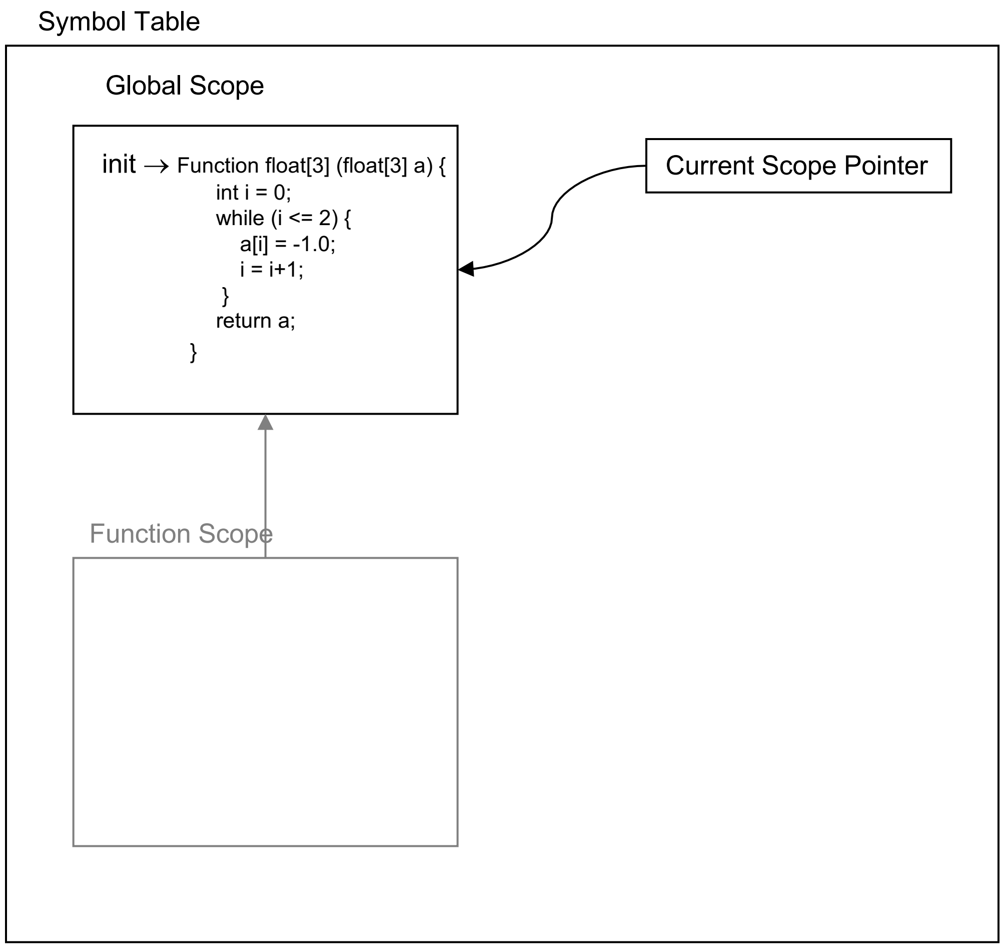
```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

Interpreting Arrays



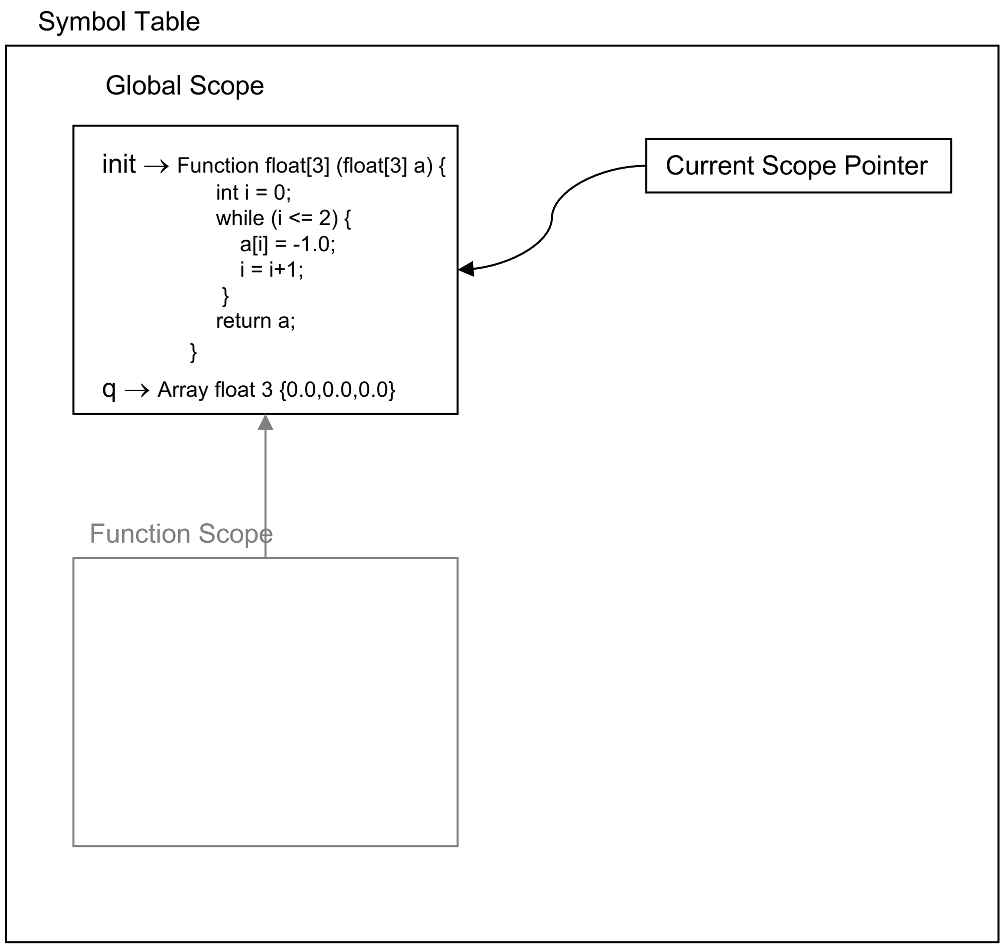
```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

Interpreting Arrays



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

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

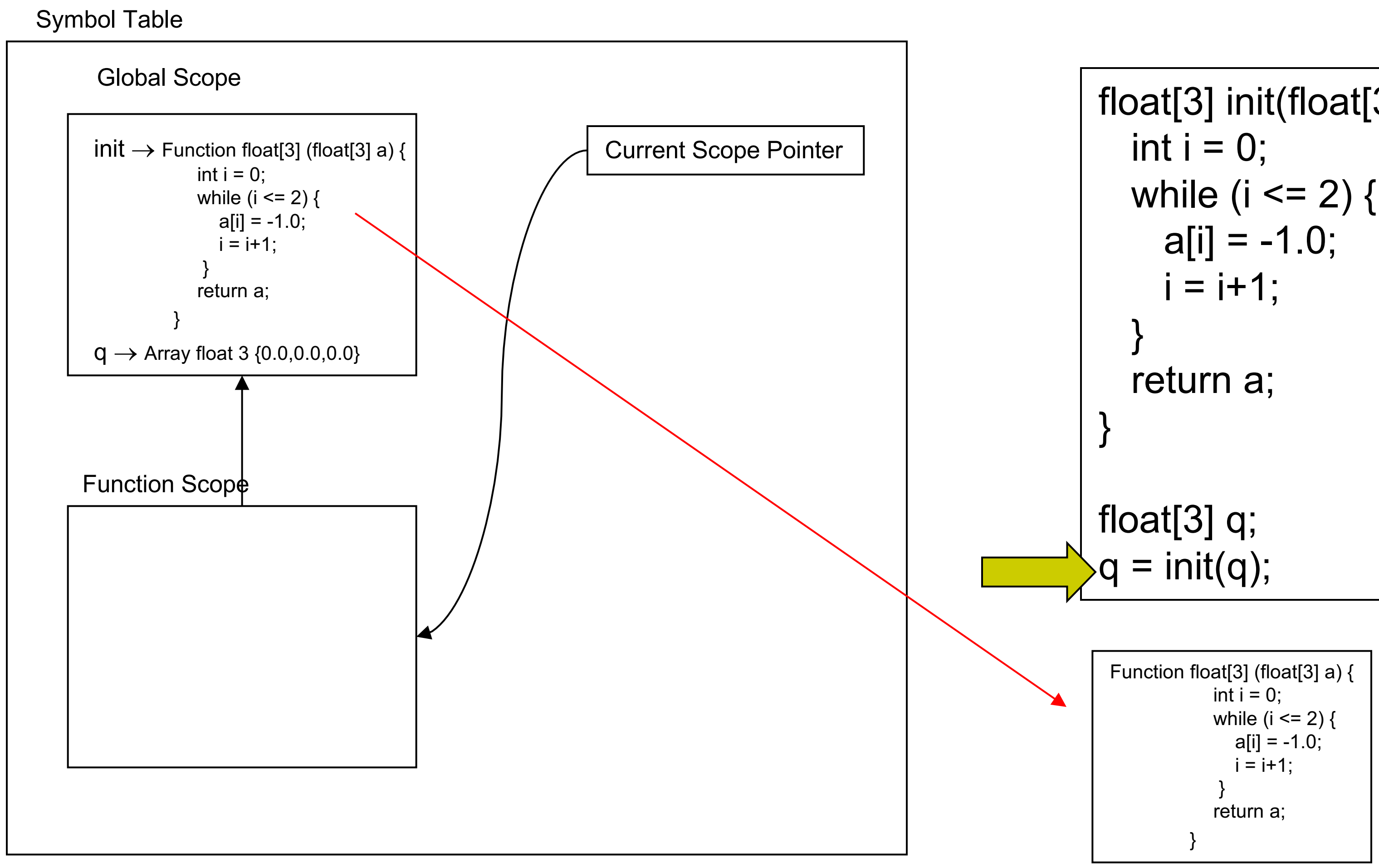
Interpreting Arrays



```
init → Function float[3] (float[3] a) {
            int i = 0;
            while (i <= 2) {
                a[i] = -1.0;
                i = i+1;
             }
            return a;
        }
q → Array float 3 {0.0,0.0,0.0}
```

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

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

Interpreting Arrays

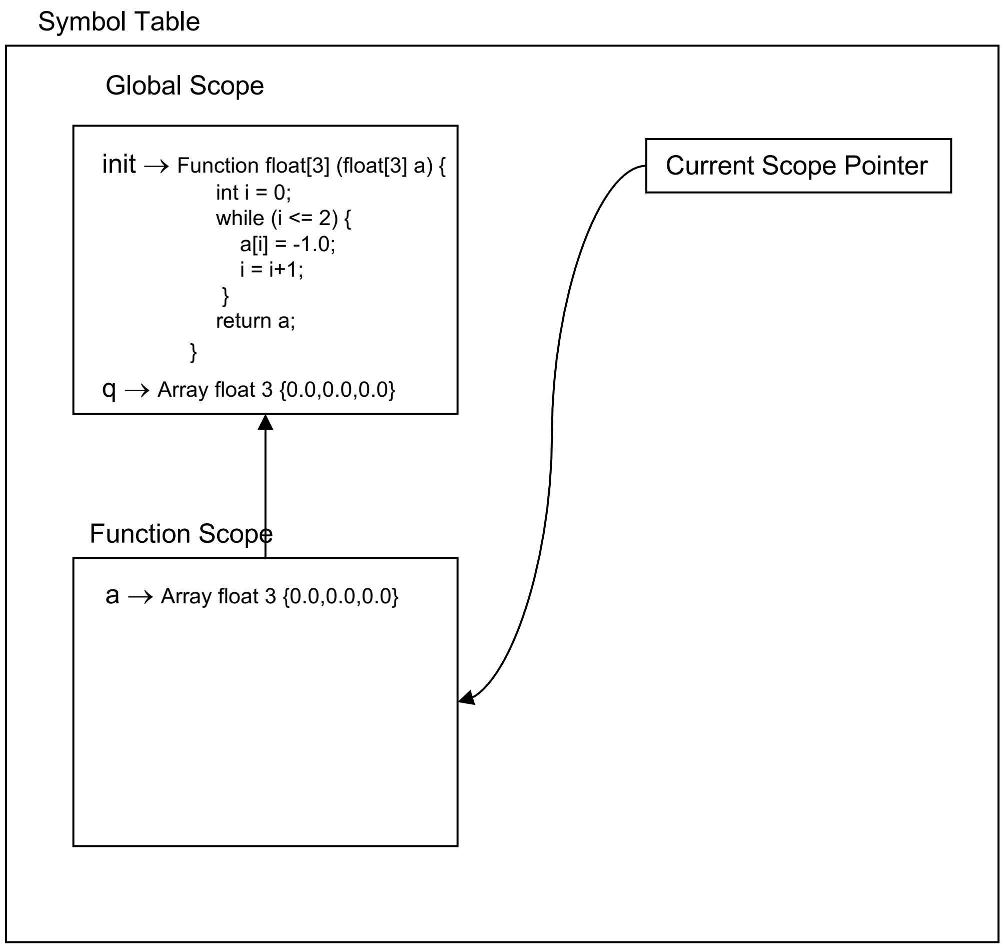


```
init → Function float[3] (float[3] a) {
           int i = 0;
           while (i <= 2) {
              a[i] = -1.0;
              i = i+1;
           }
           return a;
       }
q → Array float 3 {0.0,0.0,0.0}
```

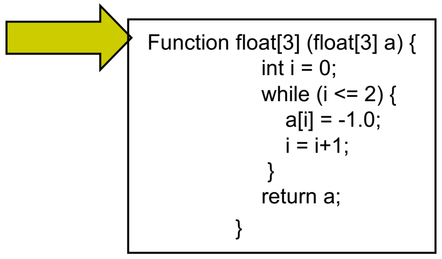
```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

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

Interpreting Arrays

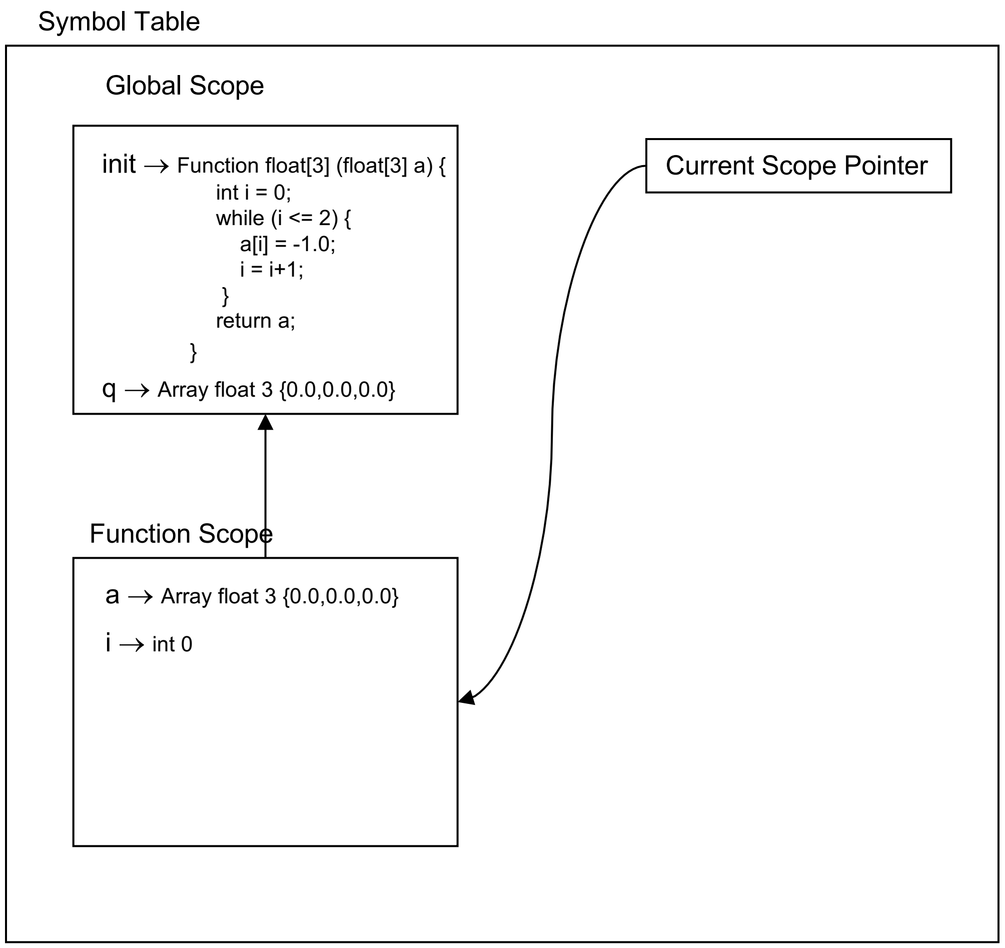


```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

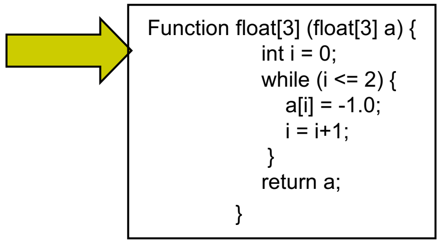


```
Function float[3] (float[3] a) {
            int i = 0;
            while (i <= 2) {
               a[i] = -1.0;
               i = i+1;
             }
            return a;
         }
```

Interpreting Arrays

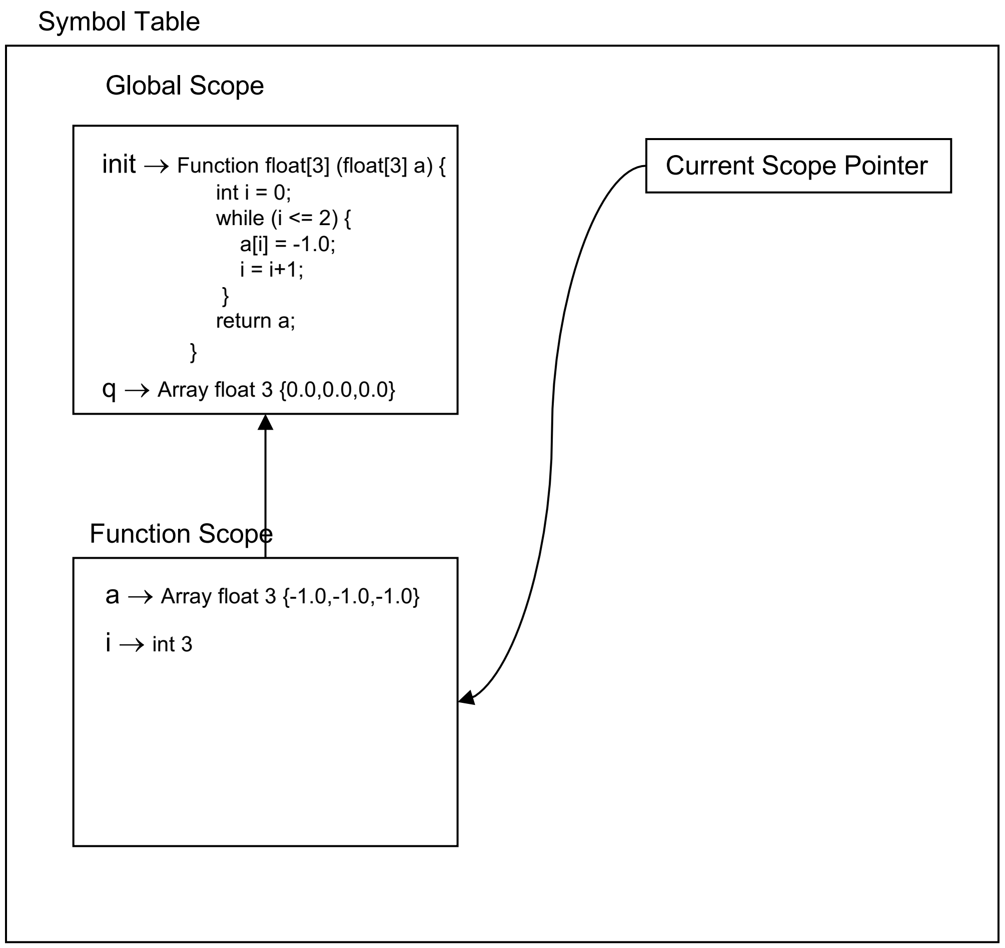


```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

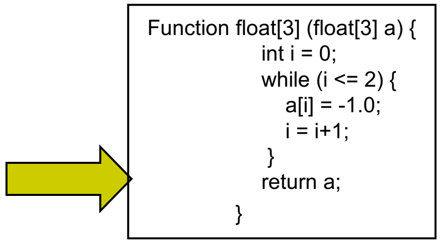


```
Function float[3] (float[3] a) {
        int i = 0;
        while (i <= 2) {
           a[i] = -1.0;
           i = i+1;
        }
        return a;
    }
```

Interpreting Arrays

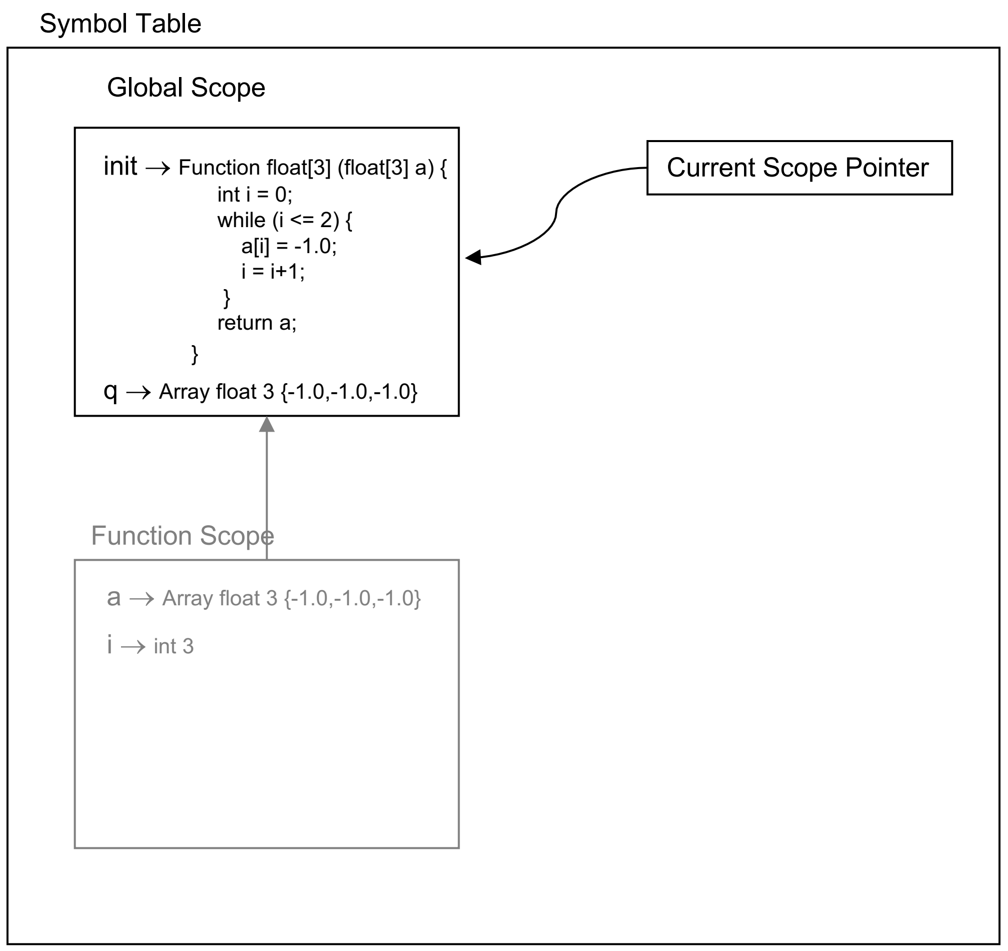


```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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



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

Interpreting Arrays



```
init → Function float[3] (float[3] a) {
           int i = 0;
           while (i <= 2) {
              a[i] = -1.0;
              i = i+1;
           }
           return a;
        }
q → Array float 3 {-1.0,-1.0,-1.0}
```

```
a → Array float 3 {-1.0,-1.0,-1.0}
i → int 3
```

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

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

Functions and Arrays

- Quicksort

```
int[100] qsort(int[100] a, int count) {
  int[100] less;
  int[100] more;
  int lesscount = 0;
  int morecount = 0;

  if (count <= 1)
    return a;

  int i = 1;
  int pivot = a[0];

  while (i <= count-1) {
    if (a[i] <= pivot) {
      less[lesscount] = a[i];
      lesscount = lesscount+1;
    }
    else {
      more[morecount] = a[i];
      morecount = morecount+1;
    }
  }

  less[lesscount] = pivot;
  lesscount = lesscount+1;

  less = qsort(less,lesscount);
  more = qsort(more,morecount);

  return append(less,lesscount,more,morecount);
}
```

Functions and Arrays

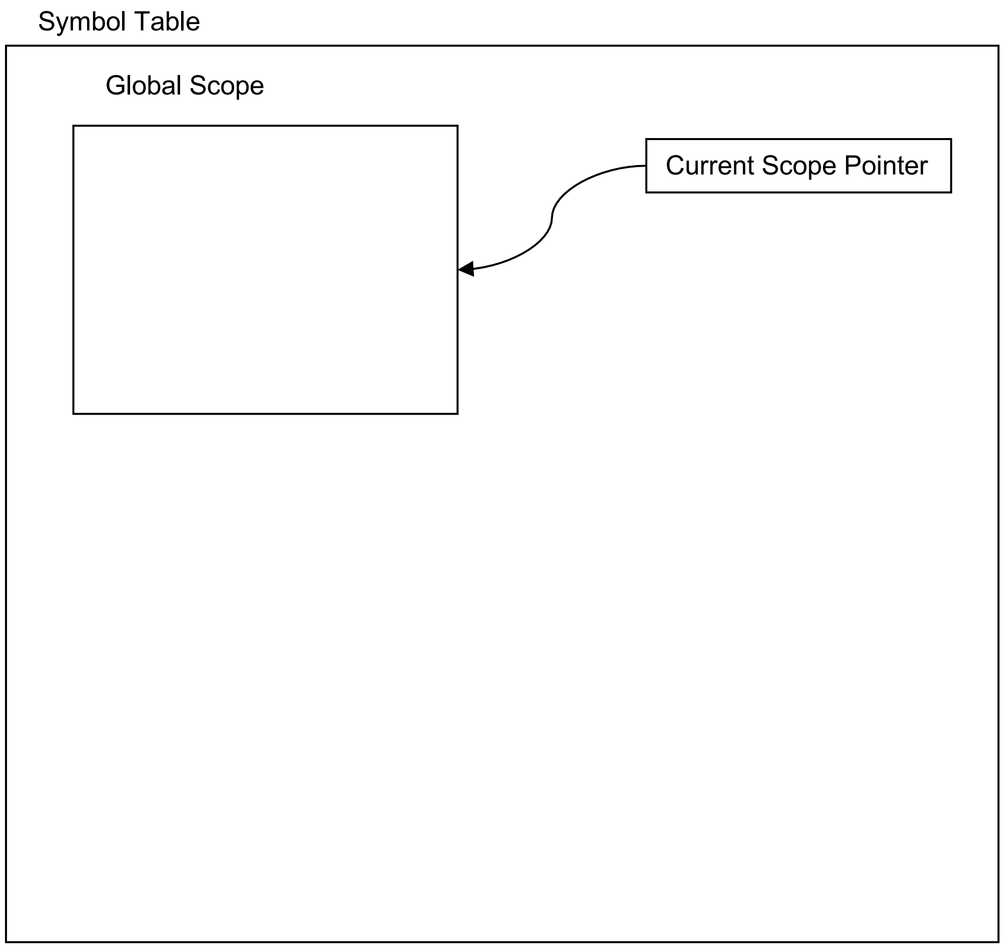
- Append

```
int[100] append(int[100] a, int acount, int[100] b, bcount) {
   int[100] result;
   int rcount = 0;
   int i = 0;

   while (i <= acount-1) {
      result[rcount] = a[i];
      rcount = rcount+1;
   }

   i = 0;
   while (i <= bcount-1) {
      result[rcount] = b[i];
      rcount = rcount+1;
   }

   return result;
}
```

References to Array Values

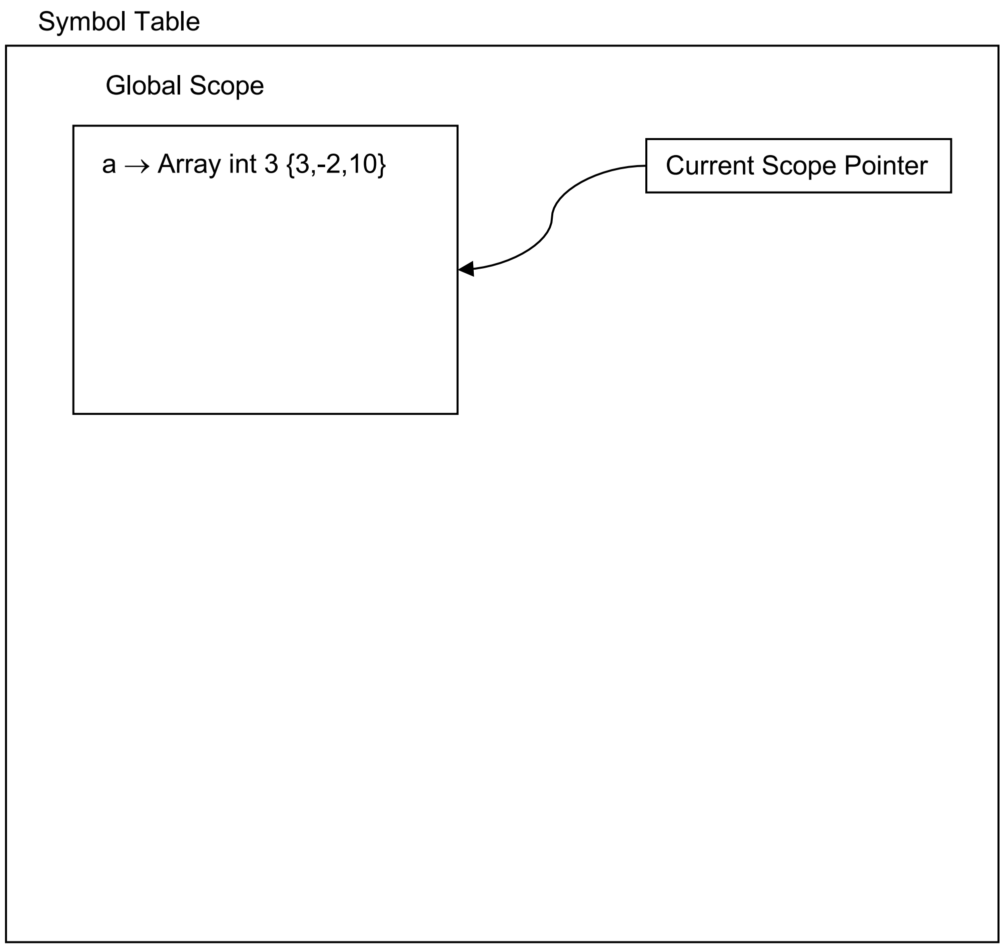
- We store references of array values in the symbol table
- When copying array values from one variable to another we copy references to the array value
- When we pass arrays to functions we pass array value references - that is with regard to array values our function calls act as call by reference

Array References



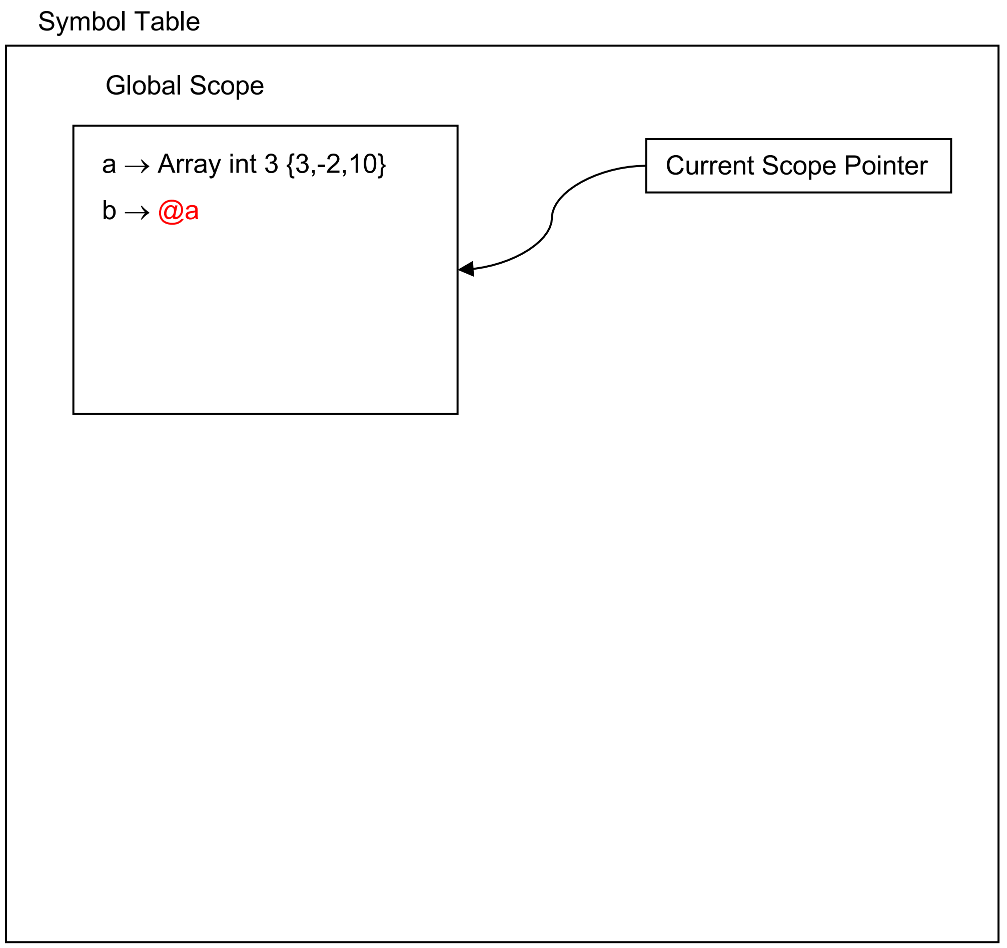
```
int[3] a = { 3,-2,10 };
int[3] b = a;
b[2] = 0;
```

Array References



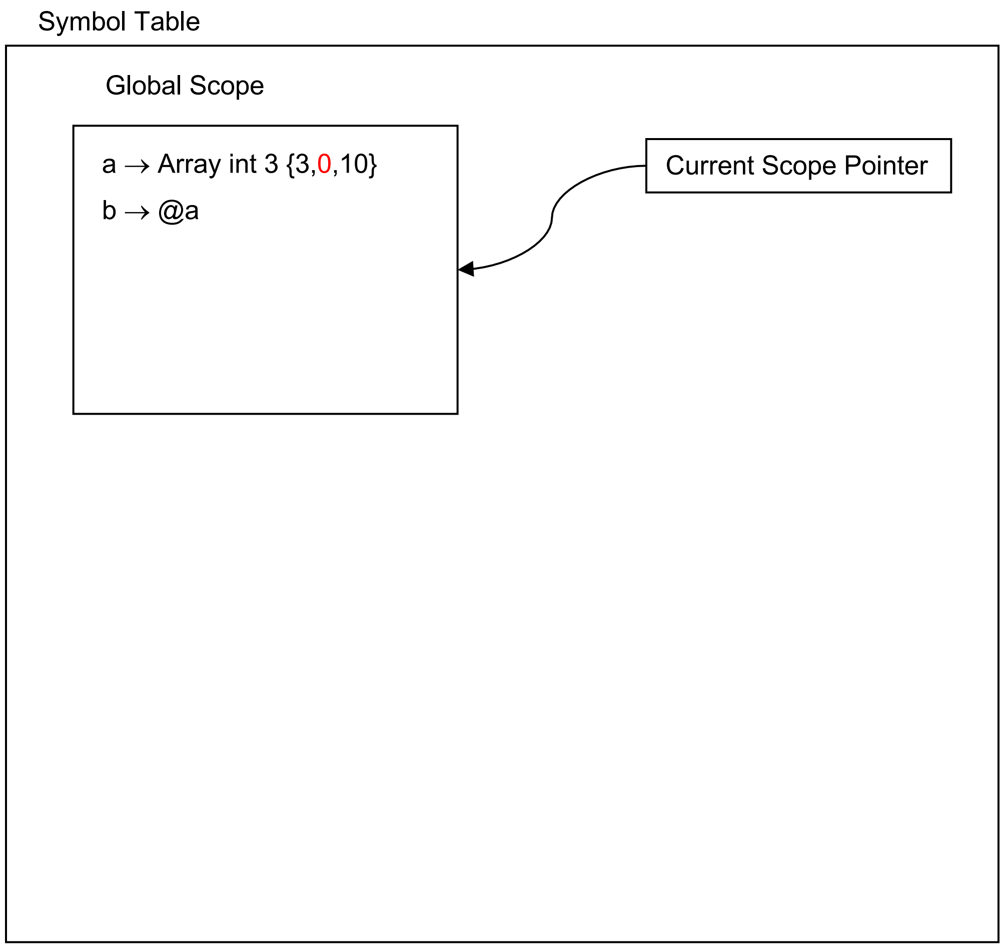
```
int[3] a = { 3,-2,10 };
int[3] b = a;
b[2] = 0;
```

Array References



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

Array References



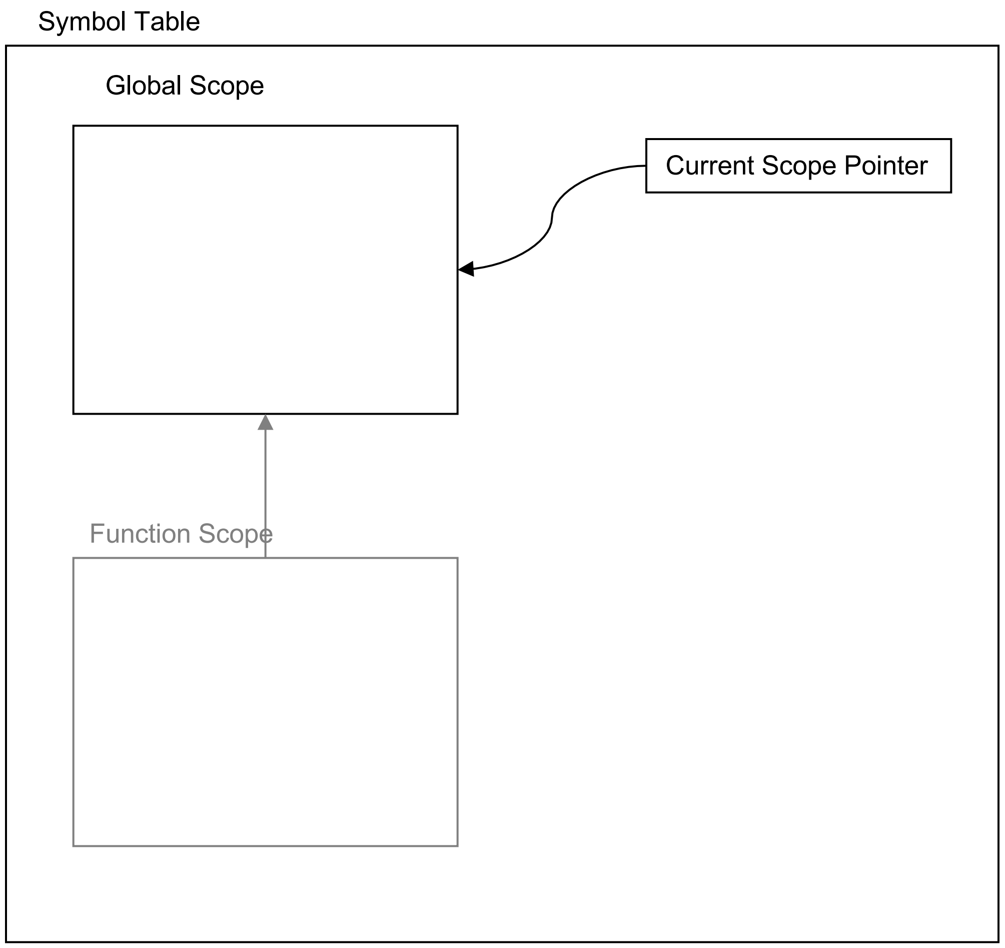
```
int[3] a = { 3,-2,10 };
int[3] b = a;
b[2] = 0;
```

References to Array Values

- A closer look at our init function reveals that the array passed in is already initialized by the loop because of the reference to the value

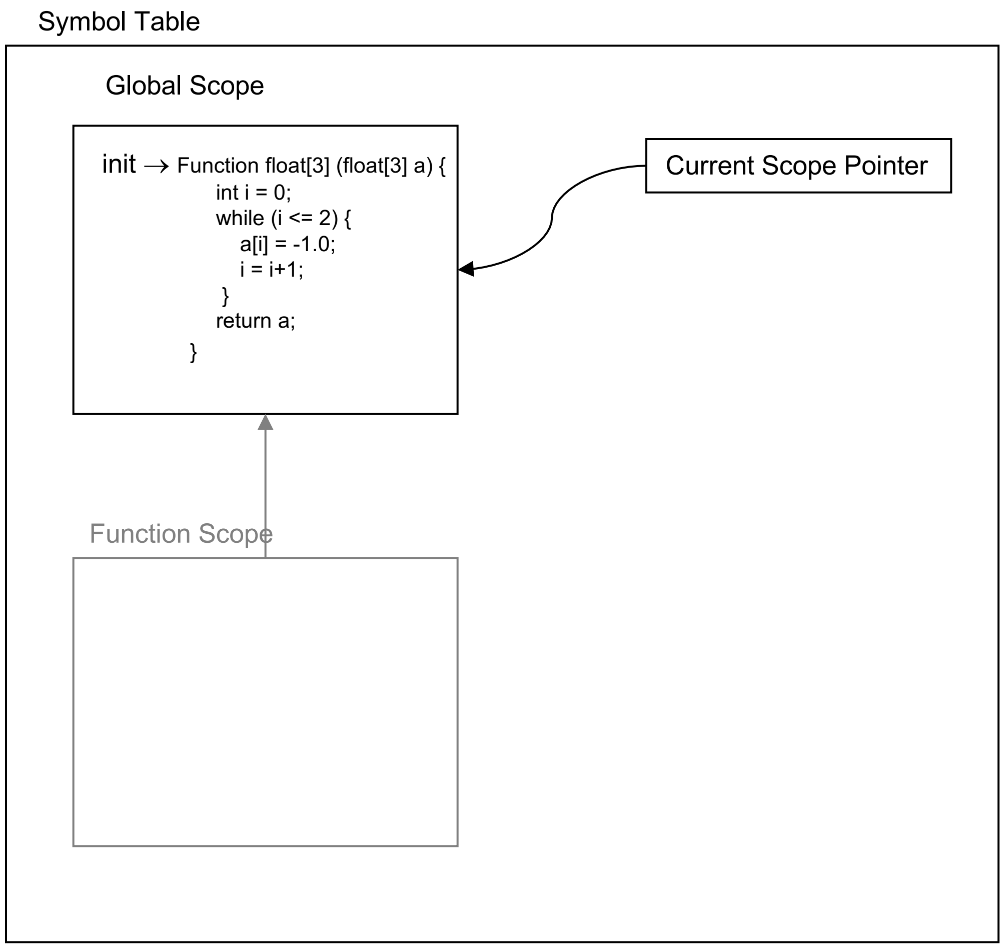
```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

Interpreting Arrays



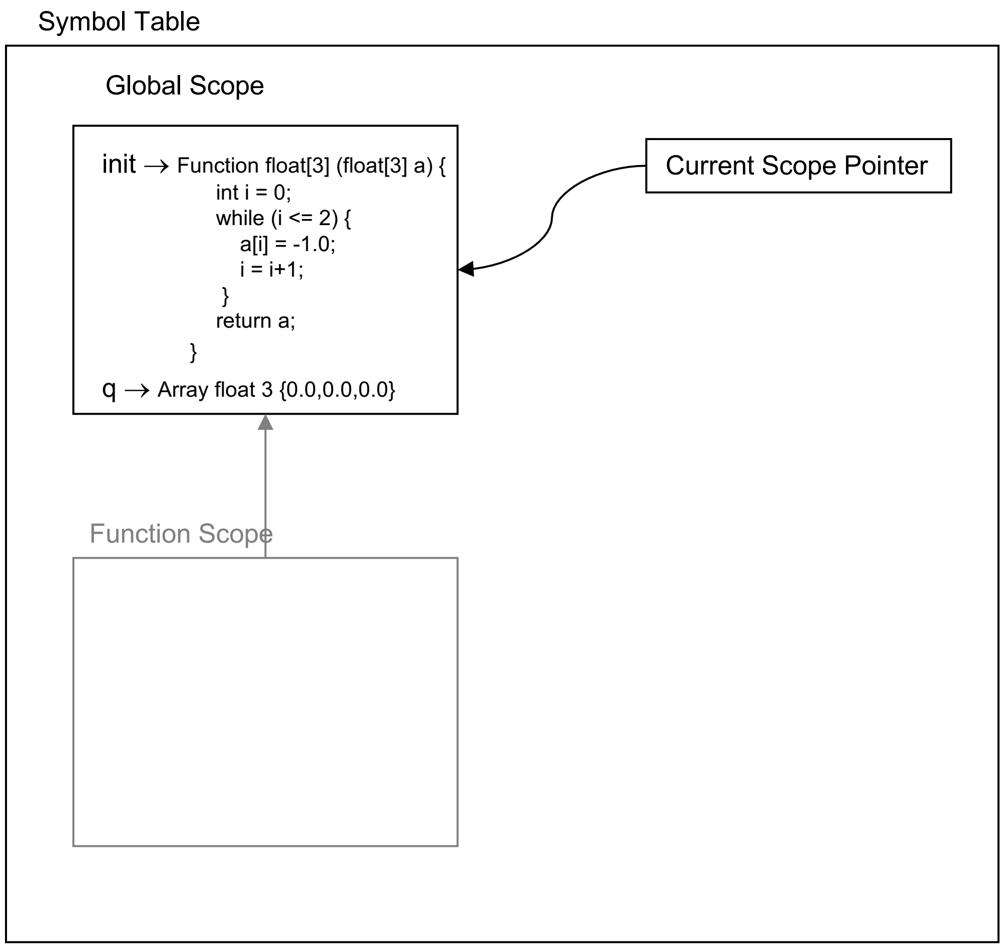
```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

Interpreting Arrays



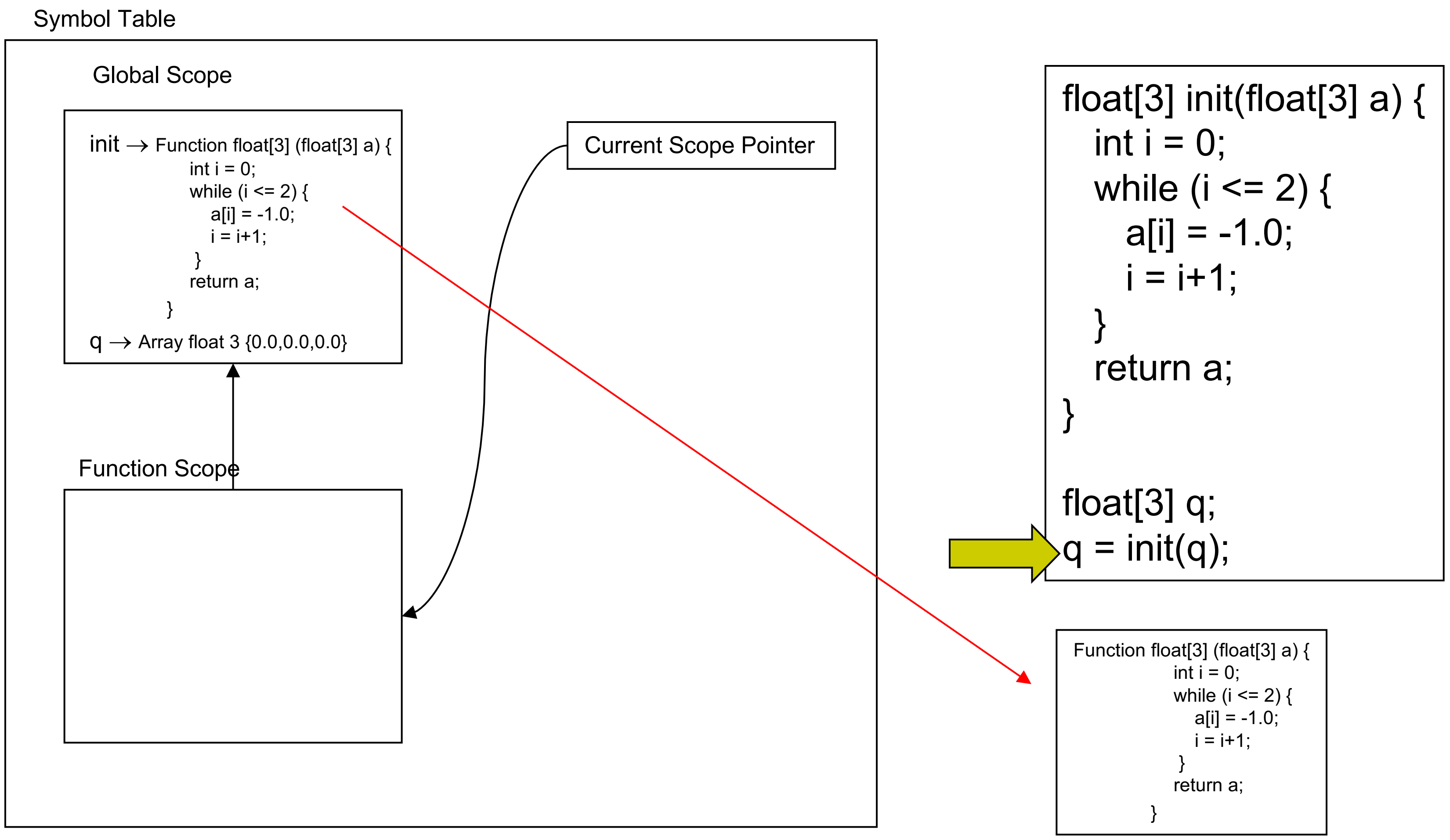
```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

Interpreting Arrays



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

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

Interpreting Arrays

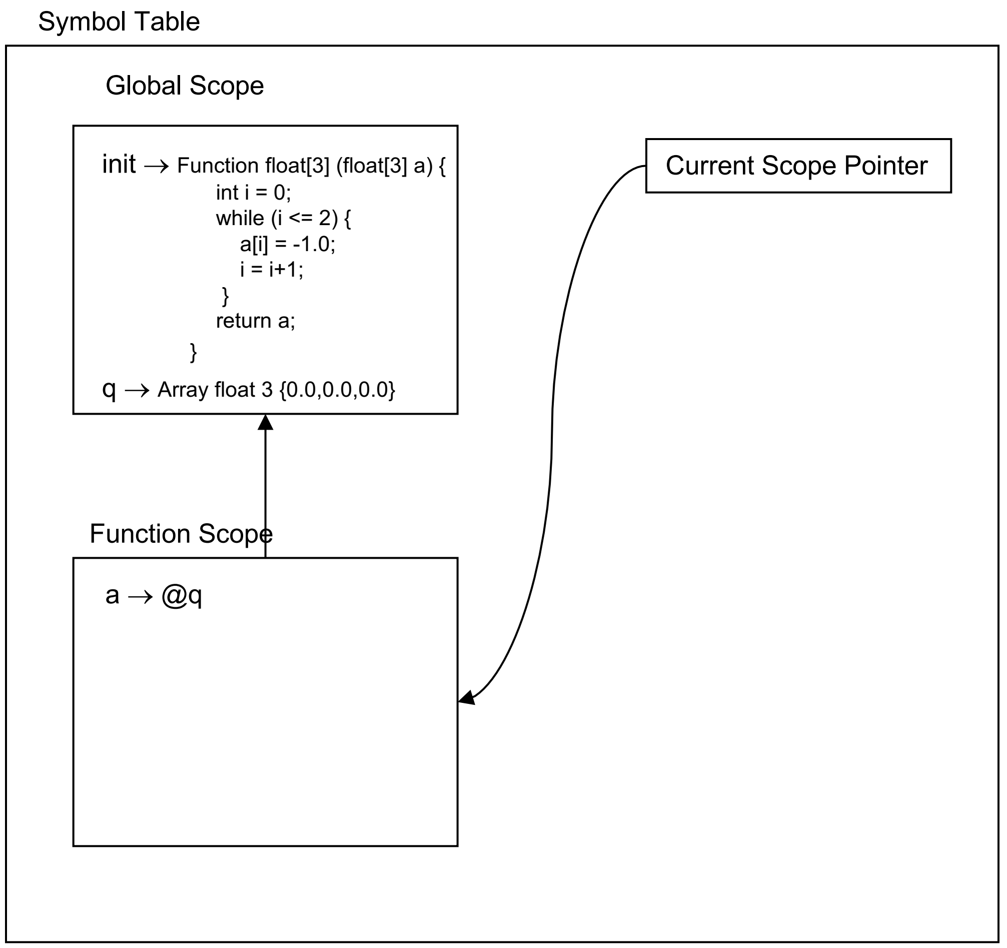


```
init → Function float[3] (float[3] a) {
           int i = 0;
           while (i <= 2) {
              a[i] = -1.0;
              i = i+1;
           }
           return a;
       }
q → Array float 3 {0.0,0.0,0.0}
```

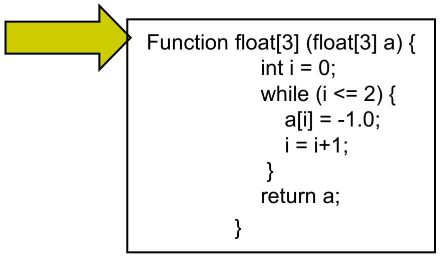
```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

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

Interpreting Arrays

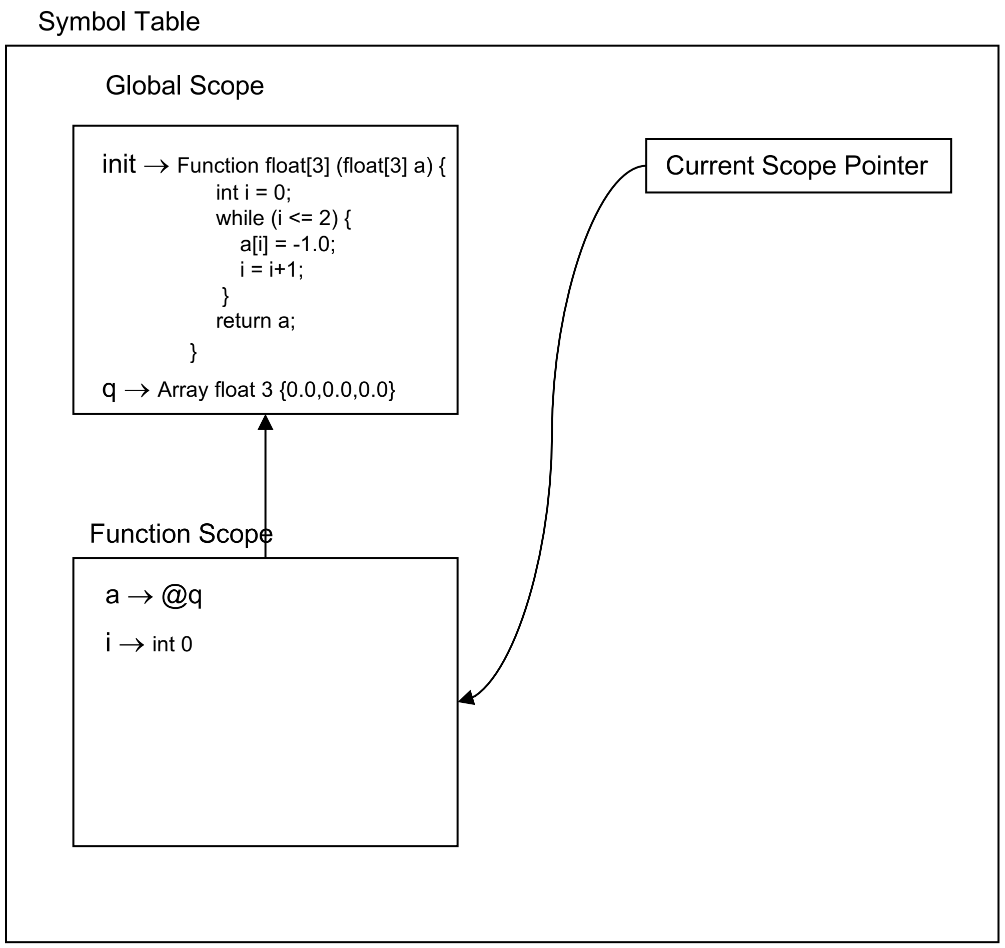


```
float[3] init(float[3] a) {
  int i = 0;
  while (i <= 2) {
    a[i] = -1.0;
    i = i+1;
  }
  return a;
}

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

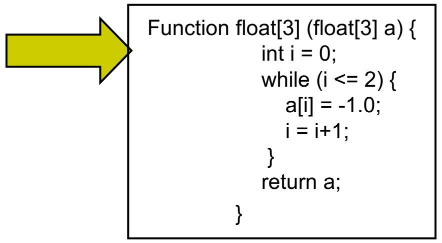


```
Function float[3] (float[3] a) {
          int i = 0;
          while (i <= 2) {
            a[i] = -1.0;
            i = i+1;
          }
          return a;
        }
```

Interpreting Arrays

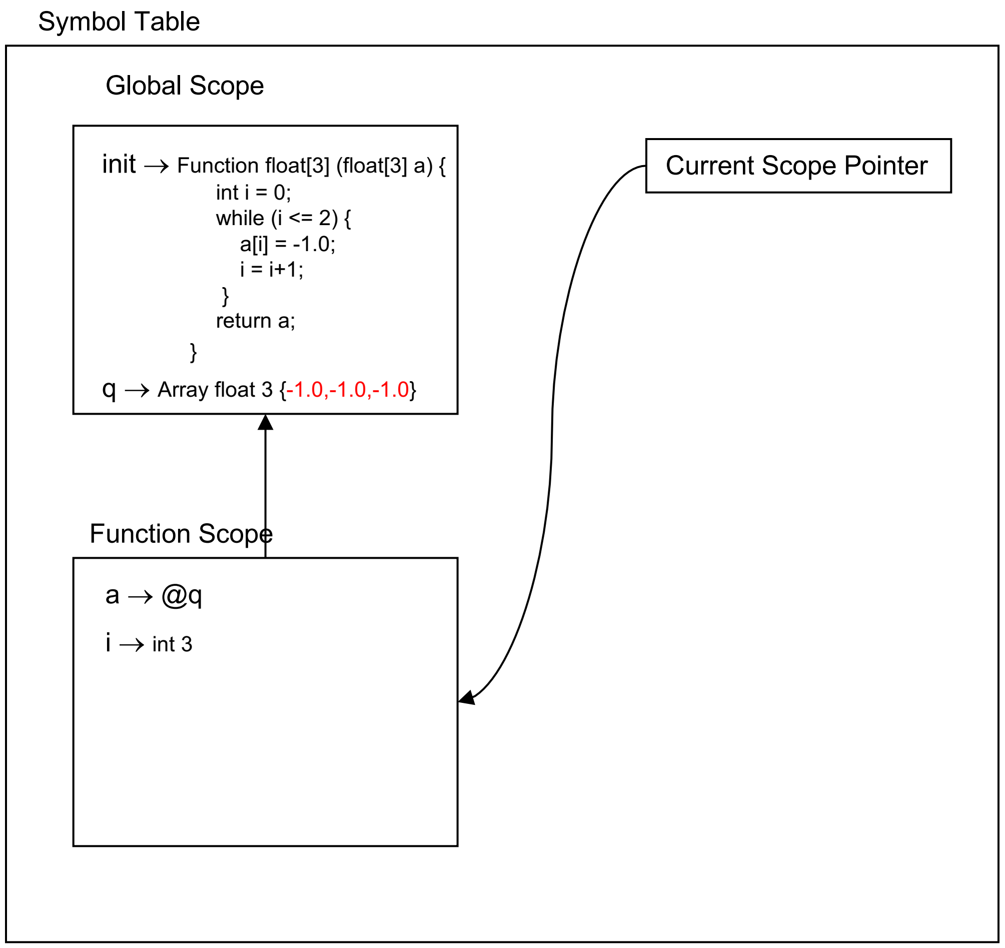


```
float[3] init(float[3] a) {
   int i = 0;
   while (i <= 2) {
      a[i] = -1.0;
      i = i+1;
   }
   return a;
}

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

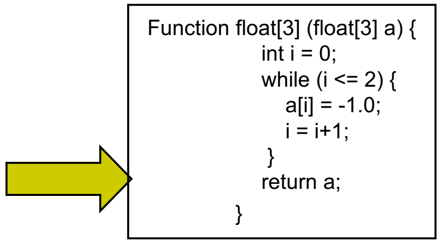


```
Function float[3] (float[3] a) {
          int i = 0;
          while (i <= 2) {
             a[i] = -1.0;
             i = i+1;
          }
          return a;
       }
```

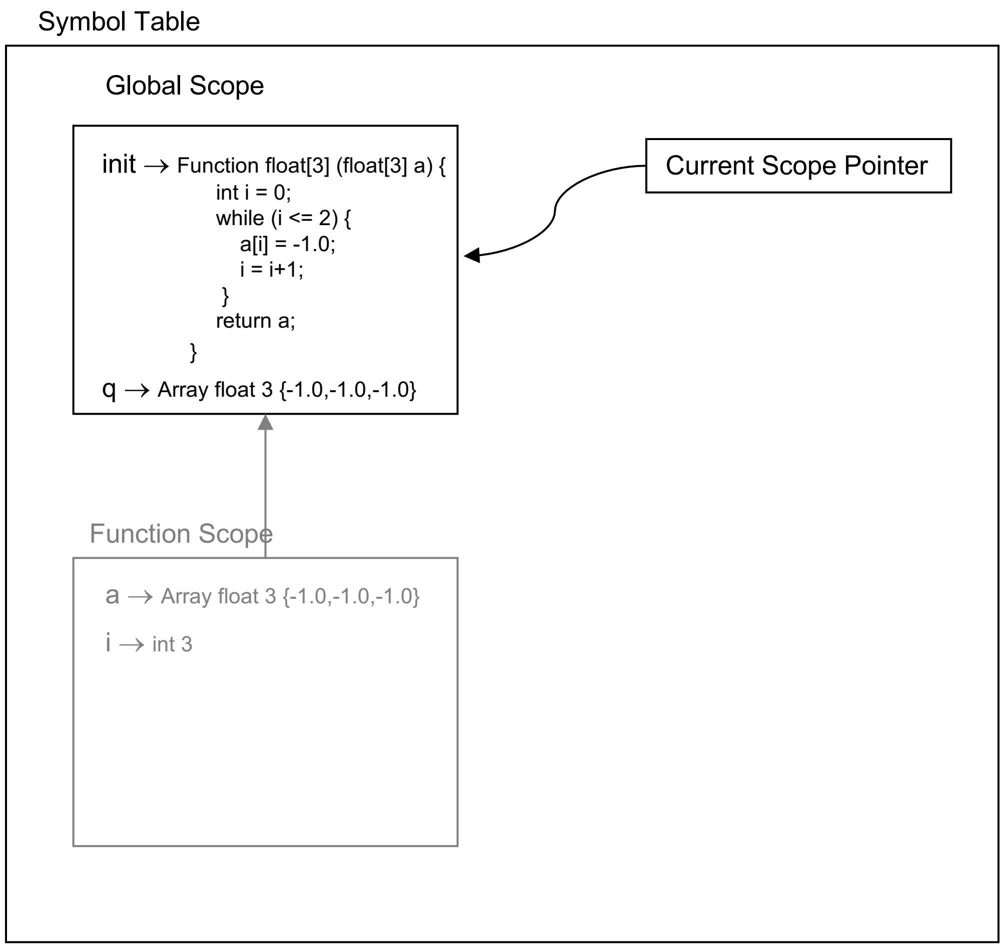
Interpreting Arrays



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

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



Interpreting Arrays



```
init → Function float[3] (float[3] a) {
           int i = 0;
           while (i <= 2) {
              a[i] = -1.0;
              i = i+1;
           }
           return a;
       }
q → Array float 3 {-1.0,-1.0,-1.0}
```

```
a → Array float 3 {-1.0,-1.0,-1.0}
i → int 3
```

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

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

References to Array Values

- A closer look at our init function reveals that the array passed in is already initialized by the loop because of the reference to the value…there is no need to pass it back.

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

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

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

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