Chapter 4

Choosing a Shape

In Chapter 3 we described how to declare a shape, which is used as a way of organizing parallel data. You can declare more than one shape in a C* program. However, a program can (in general) operate on parallel data from only one shape at a time. That shape is known as the *current* shape. You designate a shape to be the current shape by using the **with** statement, which C* has added to Standard C.

4.1 The with Statement

Assume a program contains these declarations for a shape and three parallel variables of that shape:

```
shape [16384]employees;
unsigned int:employees employee_id, age, salary;
```

Before operations can be performed on these parallel variables, **employees** must become the current shape.

To make **employees** the current shape, use the **with** statement as follows:

```
with (employees)
```

Any statement (or block of declarations and statements) following **with (employees)** can operate on parallel variables of shape **employees**. For example,

```
with (employees)
    age = 0;
```

initializes all elements of the parallel variable **age** to 0. (We discuss parallel assignment statements in Chapter 5.) If each element of **salary** has been initialized to each employee's current salary, this code:

```
unsigned int:employees new_salary;
with (employees)
    new_salary = salary*2;
```

stores twice each employee's salary in the elements of **new_salary**. (Once again, we cover arithmetic with parallel variables in the next chapter.)

You can also include operations on scalar variables inside a **with** statement. For example, you can declare a scalar variable called **sample_salary** and assign one of the values of **salary** to it:

```
with (employees) {
    unsigned int sample_salary;
    sample_salary = [0]salary;
}
```

Here is what you *can't* do inside a **with** statement:

```
shape [16384]employees, [8192]equipment;
unsigned int employee_id:employees,
            date_of_purchase:equipment;

main()
{
    with (employees)
         date_of_purchase = 0;   /* This is wrong */
}
```

The program cannot perform this operation on **date_of_purchase**, since this parallel variable is not of the current shape. However, this is legal:

```
shape [16384]employees, [8192]equipment;
unsigned int employee_id:employees,
             date_of_purchase:equipment;

main()
{
    with (employees)
         [6]date_of_purchase = 0;    /* This is legal */
}
```

In this case, **[6]date_of_purchase** is scalar, since it refers to a single element. Scalar operations are allowed on parallel variables that are not of the current shape.

See Section 4.4 for a list of the situations in which a program can operate on parallel variables that are not of the current shape.

4.1.1 Default Shape

Note that the sample program in Chapter 2 included a **with** statement, even though only one shape was declared. You must include a **with** statement to perform parallel operations on parallel data, even if only one shape has been declared.

4.1.2 Using a Shape-Valued Expression

You can use a shape-valued expression instead of a shape name to specify the current shape. For example:

```
shape [16384]employees;
unsigned int:employees age, salary;

main()
{
    with (shapeof(age))
         salary = 200;
}
```

The current shape is **employees**, because **shapeof(age)** returns the shape of the parallel variable **age**.

4.2 Nesting with Statements

Consider this **with** statement:

```
with (employees)
    add_salaries();
```

where **add_salaries** is a function defined elsewhere in the program. Clearly, **employees** remains the current shape while executing the code within **add_salaries**. But what if **add_salaries** contains its own **with** statement? The new **with** statement then takes effect, and the shape it specifies becomes current. When the **with** statement's scope is completed, **employees** once again becomes the current shape.

You can therefore nest **with** statements. The current shape is determined by following the chain of function calls to the innermost **with** statement. Returning to an outer level resets the current shape to what it was at that outer level. For example:

```
shape [16384]ShapeA, [32768]ShapeB;
int:ShapeA p1, p2;
int:ShapeB q1;

main()
{
    with (ShapeA) {
         p1 = 6;
         with (ShapeB)
              q1 = 12;
         p2 = 18;
    }
}
```

Once the code in this example leaves the scope of the nested **with** statement, **ShapeA** once again becomes the current shape. The assignment to **p2** is therefore legal.

The **break**, **goto**, **continue**, and **return** statements also reset the current shape when they branch to an outer level. For example, this code is legal:

```
with (ShapeA) {
    loop:
    /* C* code in ShapeA . . . */
    with (ShapeB) {
         /* C* code in ShapeB . . . */
         goto loop;
    }
}
```

When the **goto** statement is executed and the program returns to **loop**, **ShapeA** once again becomes the current shape.

C* does not define the behavior when a program branches *into* the body of a nested **with** statement, however. For example, this code results in undefined behavior:

```
goto loop;
    with (ShapeA) {
         loop:  /* This is wrong */
    }
```

4.3 Initializing a Variable at Block Scope

Section 3.10 described how to initialize parallel variables; it stated that you can initialize an automatic variable with an expression that can be evaluated at the variable's scope. Note that if the expression contains a parallel variable, the parallel variable must therefore be of the current shape. In the code below, **p2** is initialized to the values of **p1**; **p1** must therefore be of the current shape.

```
shape [16384]ShapeA;
int:ShapeA p1 = 6;

main()
{
    with (ShapeA) {
         int:ShapeA p2 = p1;
         /* ... */
    }
}
```

4.4 Parallel Variables Not of the Current Shape

As we mentioned above, there are certain situations in which a program can operate on a parallel variable that is not of the current shape. They are as follows:

- You can declare a parallel variable of a shape that is not the current shape. You cannot initialize the parallel variable using another parallel variable, however (because that involves performing an operation on the parallel variable being declared).

- As we discussed in Section 4.1, a parallel variable that is not of the current shape can be operated on if it is left-indexed by a scalar or scalars, because it is treated as a scalar variable.
- You can left-index any valid C* expression with a parallel variable of the current shape, in order to produce an lvalue or rvalue of the current shape. This topic is discussed in detail in Chapter 10.
- You can apply an intrinsic function like **`dimof`** and **`shapeof`** to a parallel variable that is not of the current shape.
- You can use the **`&`** operator to take the address of a parallel variable that is not of the current shape. See Chapter 7.
- You can right-index a parallel array that is not of the current shape with a scalar expression.
- You can use the “dot” operator to select a field of a parallel structure or union that is not of the current shape — provided that the field is not an aggregate type (for example, another structure or union).

You can also perform these operations (except for left-indexing by a parallel variable) even if there is *no* current shape — that is, outside the scope of any **`with`** statement.