ARRAYS

ARRAYS

- Array: a collection of a fixed number of components wherein all of the components have the same data type
- In a one-dimensional array, the components are arranged in a list form
- Syntax for declaring a one-dimensional array:

```
dataType arrayName[intExp];
```

`intExp` evaluates to a positive integer

ARRAYS (CONTINUED)

- Example:

```
int num[5];
```

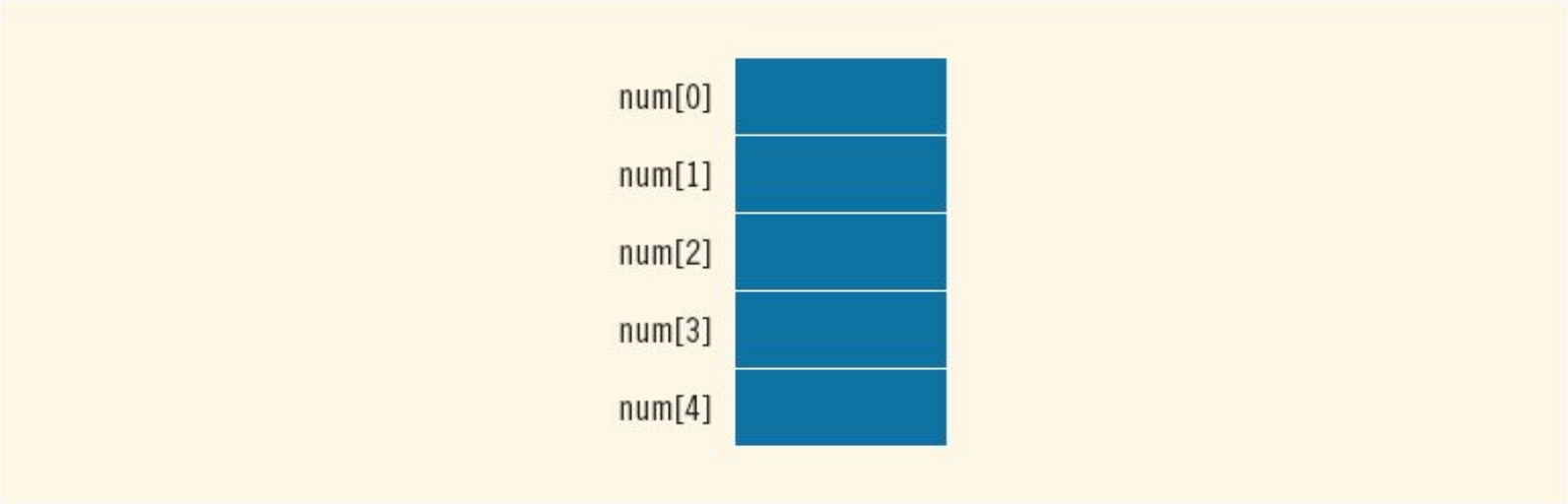

FIGURE 9-1 Array `num`

ACCESSING ARRAY COMPONENTS

- General syntax:

```
arrayName[indexExp]
```

  where `indexExp`, called an **index**, is any expression whose value is a nonnegative integer
- Index value specifies the position of the component in the array
- `[]` is the **array subscripting operator**
- The array index always starts at `0`

ACCESSING ARRAY COMPONENTS (CONTINUED)

```
int list[10];
```

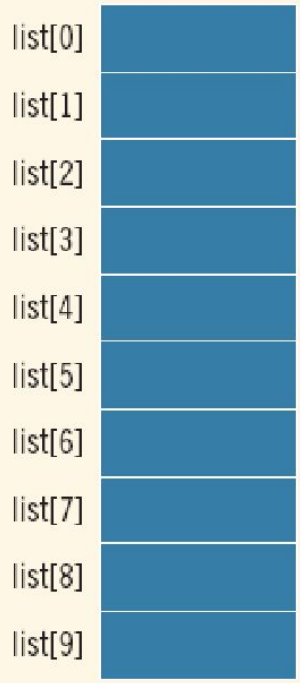

FIGURE 9-2 Array `list`

ACCESSING ARRAY COMPONENTS (CONTINUED)

```
list[5] = 34;
```

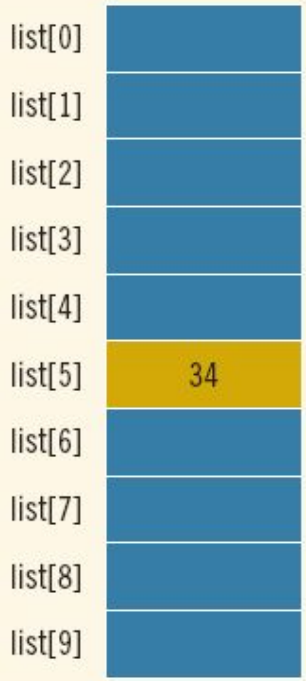

FIGURE 9-3 Array `list` after execution of the statement `list[5]= 34;`

ACCESSING ARRAY COMPONENTS (CONTINUED)

```
list[3] = 10;
list[6] = 35;
list[5] = list[3] + list[6];
```

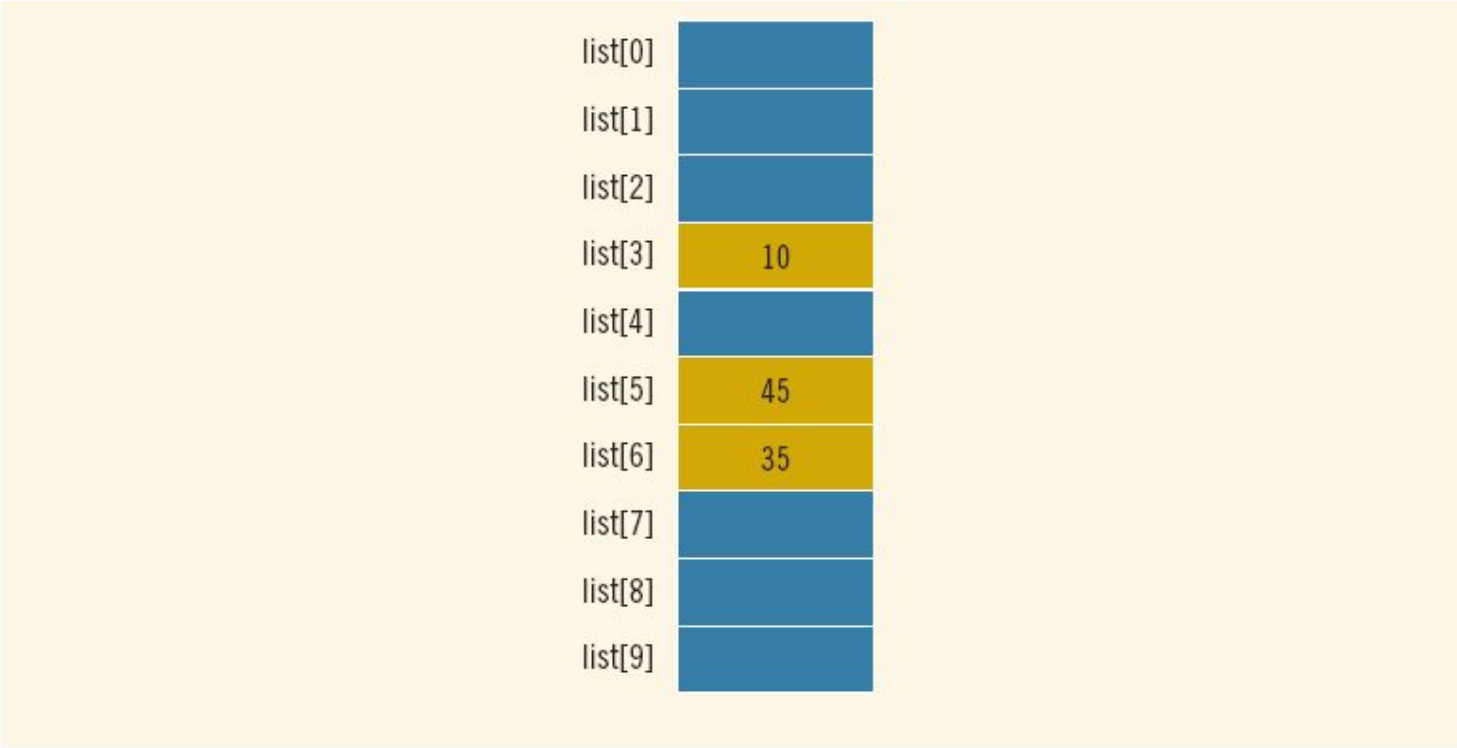

FIGURE 9-4 Array `list` after execution of the statements `list[3]= 10;`, `list[6]= 35;`, and `list[5] = list[3] + list[6];`

ACCESSING ARRAY COMPONENTS (CONTINUED)

You can also declare arrays as follows:

```
const int ARRAY_SIZE = 10;
int list[ARRAY_SIZE];
```

That is, you can first declare a named constant and then use the value of the named constant to declare an array and specify its size.

PROCESSING ONE-DIMENSIONAL ARRAYS

- Some basic operations performed on a one-dimensional array are:
  - Initializing
  - Inputting data
  - Outputting data stored in an array
  - Finding the largest and/or smallest element
- Each operation requires ability to step through the elements of the array
- Easily accomplished by a loop

PROCESSING ONE-DIMENSIONAL ARRAYS (CONTINUED)

- Consider the declaration

```
int list[100];  //array of size 100
int i;
```

- Using `for` loops to access array elements:

```
for (i = 0; i < 100; i++)    //Line 1
        //process list[i]  //Line 2
```

- Example:

```
for (i = 0; i < 100; i++)    //Line 1
     cin >> list[i];    //Line 2
```

ARRAY INDEX OUT OF BOUNDS

- If we have the statements:

```
double num[10];
int i;
```

- The component `num[i]` is valid if `i = 0, 1, 2, 3, 4, 5, 6, 7, 8,` or `9`
- The index of an array is in bounds if the `index >=0` and the `index <= ARRAY_SIZE-1`
  - Otherwise, we say the `index` is out of bounds
- In C++, there is no guard against indices that are out of bounds

PARTIAL INITIALIZATION OF ARRAYS DURING DECLARATION

- The statement:

```
int list[10] = {0};
```

  declares `list` to be an array of `10` components and initializes all of them to zero

- The statement:

```
int list[10] = {8, 5, 12};
```

  declares `list` to be an array of `10` components, initializes `list[0]` to `8`, `list[1]` to `5`, `list[2]` to `12` and all other components are initialized to `0`

PARTIAL INITIALIZATION OF ARRAYS DURING DECLARATION (CONTINUED)

- The statement:

```
int list[] = {5, 6, 3};
```

  declares `list` to be an array of `3` components and initializes `list[0]` to `5`, `list[1]` to `6`, and `list[2]` to `3`

- The statement:

```
int list[25]= {4, 7};
```

  declares an array of `25` components; initializes `list[0]` to `4` and `list[1]` to `7`; all other components are initialized to `0`

SOME RESTRICTIONS ON ARRAY PROCESSING

- Consider the following statements:

```
int myList[5] = {0, 4, 8, 12, 16};  //Line 1
int yourList[5];  //Line 2
```

- C++ does not allow aggregate operations on an array:

```
yourList = myList;  //illegal
```

- Solution:

```
for (int index = 0; index < 5; index ++)
     yourList[index] = myList[index];
```

SOME RESTRICTIONS ON ARRAY PROCESSING (CONTINUED)

- The following is illegal too:

```
cin >> yourList; //illegal
```

- Solution:

```
for (int index = 0; index < 5; index ++)
    cin >> yourList[index];
```

- The following statements are legal, but do not give the desired results:

```
cout << yourList;

if (myList <= yourList)
.
.
```

```
#include <iostream>
using namespace std;
int main()
{
int marks[10];
//input data
for(int i=0;i<10;i++)
{
cout<<"enter marks["<<i<<"]"<<endl;
cin>>marks[i];
}
// output data
for(int j=0;j<10;j++)
{

cout<<"marks["<<j<<"]= "<<marks[j]<<endl;

}
return 0;
}
```

```
#include <iostream>
using namespace std;
void printArray(int[]);
int main()
{
int marks[10];
//input data
for(int i=0;i<10;i++)
{
cout<<"enter marks["<<i<<"]"<<endl;
cin>>marks[i];
}
printArray(marks);
return 0;
}
void printArray(int marks[])
{
// output data
for(int j=0;j<10;j++)
{
cout<<"marks["<<j<<"]= "<<marks[j]<<endl;
}
}
```

```
#include <iostream>
using namespace std;
void printArray(int[]);
void inputArray(int[],int);
int main()
{
int marks[10];
inputArray(marks,10);
printArray(marks);
return 0;
}

void inputArray(int marks[],int size)
{
//input data
for(int i=0;i<size;i++)
{
cout<<"enter marks["<<i<<"]"<<endl;
cin>>marks[i];
}
}
void printArray(int marks[])
{
// output data
for(int j=0;j<10;j++)
cout<<"marks["<<j<<"]= "<<marks[j]<<endl;
{
}
}
```