1. Definitions

We will define several terms that you need to know to understand arrays. They are as follows:

a. The name of a **static array** is a **constant** pointer to the first element in the array.
b. An array is static if its size/capacity must be known a compile time.
c. The **size/capacity** is the number of memory cells allocated to an array. Also referred to as the declared size.
d. The data type of an array is referred to as the base type or type.
e. All the elements in an array have the same base type. Thus, an array is referred to as a homogeneous data type.
f. An **index/subscript** is used to access the memory cells in an array.
g. The elements of an array are referred to as subscripted variables or elements.
h. [] is called the subscript operator.
i. The **index** is ALWAYS a non-negative integer.
j. The **range** of an index is between O and the size-1.

2. Array Declarations

a. syntax: type_name Array_Name[Declared_size];
b. Examples:
    - int score[S]; // example of an array of integer
    - call_record call database[12]; //example of an array of records
    - string page[l00]; //example of an array of strings

3. Using Constants to help define an Array

a. consider the following:

```
const int NUMBER_OF _STUDENTS = 40;          //remember, constants all
                                             //defined with capital letters
scores[NUMBER_OF_STUDENTS];
```

4. Arrays in Memory

a. two parts:
    - base address - this is the address of the first item in the array. For example, score[O], or call_database[0].

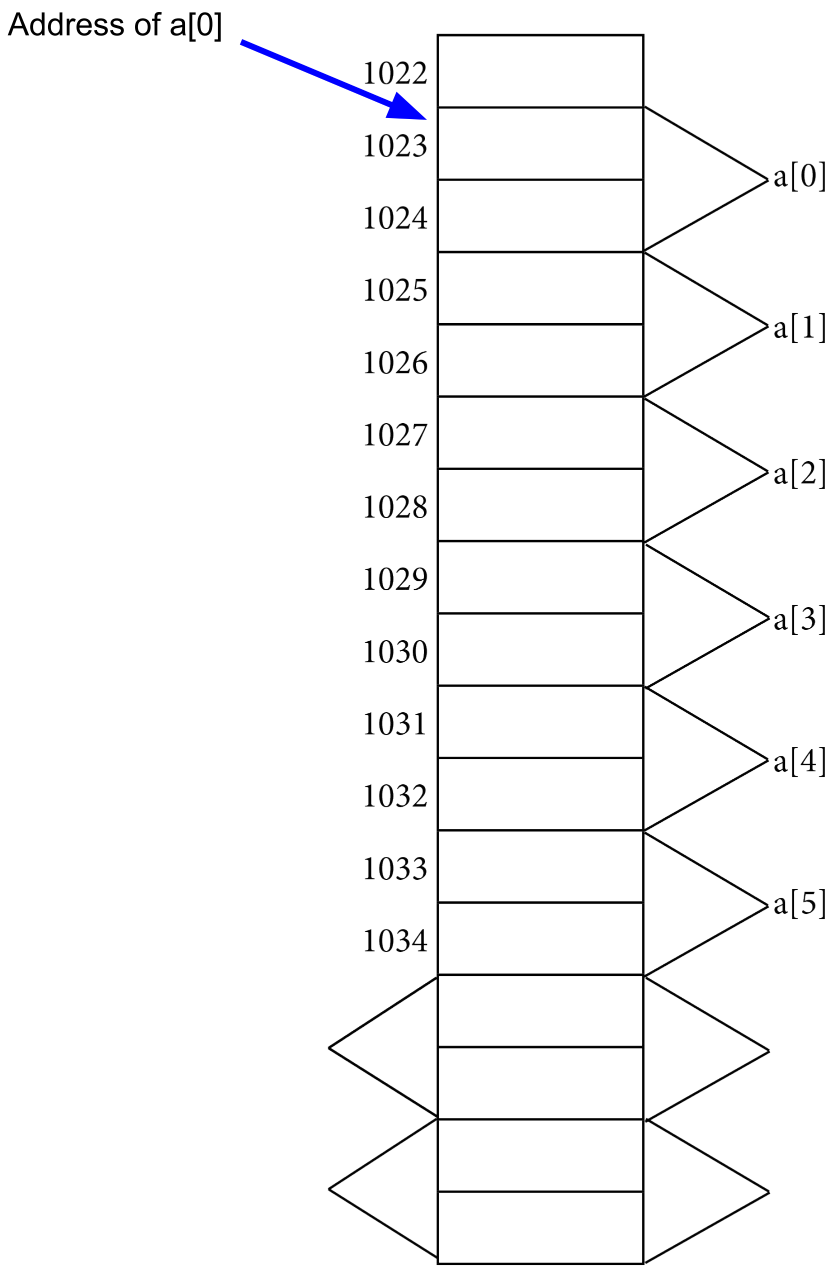
b. See drawing on next page:

An Array “a” in memory

- a is a constant pointer to 1st element in the array.
- Where is a [6]

6 x 2 = 12 bytes from a [0]

12 + 1023 = 1035



5. Pitfall

a. You must make sure that the array index is between 0 and the SIZE-1. If not, your array will be out of range.
b. Out of range errors cause you program to behave unpredictable.
c. You always remember to check that 0 <=index< SIZE is true.

6. Initializing an Array

a. Initialize when declared as in the following examples:
   - int children[3] = {2,12,1};
   - char alpha[2] = {'a','b'};
   - string names[2] ={"Lofton", "Bullard"};

b. Initialize each element one by one with separate assignment statements as in the following examples:
   - example 1:
     1. int children[3];
        children[0] =2;
        children[l] = 12;
        children(2] = 1;
     2. int alpha[2];
        alpha[0] = **'a'**;
        alpha[l] = 'b';
     3. string names[2];
        names[0] = "Lofton";
        names[l] = "Bullard";

c. Initialize with a for loop as in the following:

```
int scores[lO0];
for(int i=0; i< 100; i++)
{
        scores[i] = 0;
}
```

7. Array in Functions

a. You may pass an array element (indexed variable) as an argument to a function. Consider the following:

```
int Total_Points(int x, int y)
{
        return x + y;
}
int main()
{
```

```
int scores [ 2 ] = {88, 42};
cout << Total_Points(scores[O], scores[l]) << endl; return
O;
}
//Note: scores is an array and scores is a constant pointer to the first
//element in the array. Each element in the memory pointed to by
//scores is an integer.
```

8. Array in Functions {being passed as parameters

a. Arrays are passed to functions through the call-by-reference mechanism.
b. Arrays are passed by reference by default.
c. The argument in the call statement is called an array argument.
d. The argument in the prototype is called an array parameter. Consider the following code examples:

Example 1:

```
void Array_Passing( int myArray [],int count) //passing an array- call-
{                                                 //by-reference
    for (int i==0; i<count; i++)
    {
            cout<< myArray[i] << endl;
    }
}
int main()
{
    int myArray[l0] = {1,2,3,4,5,6,7,8,9,10};
    Array_Passing(myArray, 10); //passing an array
    return 0;
}
```

Example 2: When the parameter is specified as a canst, then it may not be changed inside the function. This means that it may not appear on the left-hand side of an assignment statement or as an argument to a function that will try to change it.

```
void Array_Passing( canst int myArray [ ], int count) //passing an array
{                                               //as a constant parameter

for (int i=O; i<count; i++)
{
cout<< myArray[i] << endl;

/ /myArray[i] = 15; / /This will not be allowing because of canst
}
}

int main()
{
int myArray[lO] = {l,2,3,4,5,6,7,8,9,10};

Array_Passing(myArray, 10); //passing an array

return O;
}
```