

Arrays in C++

Instructor: Andy Abreu

Reason behind the idea

- When we are programming, often we have to process a large amount of information. We can do so by creating a lot of variables to keep track of them.
- However this approach is not the best. Why not?

Arrays

- Arrays are great for keeping track of similar group of data.
- What are some scenarios where using arrays can help us?

Declaring an Array

- Model:

type_of_array name_of_array [size_of_array]

type_of_array: The data type, example: int

*name_of_array: The name of the array,
example: grades*

size_of_array: The capacity of the array, example: 10

- Examples:

- `int grades[10];`

- `string students[10];`

Accessing the entire Array

- If we have the following array declared:
 - `int grades[10];`
- To access the entire array we would refer to `grades`.
- For example if we want to pass the array into a function, we would pass **grades into the function as** an argument.

Accessing Elements in the Array

- If we have an array declare as the following:
 - `int grades[5];`
- The elements of the array are as follows:
 - `grades[0]`
 - `grades[1]`
 - `grades[2]`
 - `grades[3]`
 - `grades[4]`
- Counting in the array starts from 0, and the last element is size – 1.

Accessing Elements in the Array

- We can assign values to the elements as follows:
 - `grades[0] = 89;`
 - `grades[1] = 93;`
 - `grades[2] = 45;`
 - `grades[3] = 78;`
 - `grades[4] = 101;`

Printing elements of the array

- Using the same array as before 'grades', we can create the following cout statements:
 - `cout << grades[0];`
 - `cout << grades[1];`
 - `cout << grades[2];`
 - `cout << grades[3];`
 - `cout << grades[4];`
- NOTE: `cout << grades` //does NOT work!
- Try it out and note what happens.

Loops and Arrays

- We can use a for loop to printing out elements of the array, code would look like this:

```
for ( int i = 0 ; i < 5 ; ++i )  
    cout << grades[i] << endl;
```

Note

- If our array is:
 - `string names[10];`
 - **names refers to the array, the whole array**
 - `names[0]` refers to the very first element
 - `names[1]` refers to the second element
 - ...
 - `names[9]` refers to the last element
 - Referring to `names[10]` will crash your program!!

Initializing the array

- Sometimes we want to pre-initialize the array, we can do the following:
 - `int lookup[5] = { 100, 90, 80, 70, 60 };`
 - `int lookup[] = { 100, 90, 80, 70, 60 };`
 - This would also work
- Sometimes we want to initialize the entire array to zero, we can do the following:
 - `int sums[10] = {0};`
 - `{0}` is a special code to C++, `{1}` doesn't work.

Initializing the array – the catch

- You will not be able to initialize arrays if the arrays size are specified by user input. So, the following will **NOT** work:

```
int x;
```

```
cin >> x;
```

```
int a[x];
```

Arrays and Functions

- Just like regular variables, arrays can be passed into functions.
- When passing arrays into functions, consider this first:
 - Pass the entire arrays into the sub function,
or
 - If only one of the element is needed, pass just that one element into the function.

Examples of passing single element

```
int main()
{
    int grades[5];
    //do something that read in grades...
    //isPassing returns “pass” or “fail”
    cout << getPassFail( grades[0] );
}
```

What does the function look like?

```
string getPassFail( int grade )  
{  
    if ( grade >= 75 )  
        return "pass";  
    return "fail";  
}
```

Example of passing entire array

```
int main()
{
    int grades[5];
    //do something that read in grades...
    printPassOrFail( grades, 5 );
}
```


What does this function look like?

```
void printPassOrFail( int grades[], int size )
{
    for ( int i = 0 ; i < size ; ++ i )
    {
        if ( grades[i] >= 75 )
            cout << grades[i] << " is passing.\n";
        else
            cout << grades[i] << " is failing.\n";
    }
}
```

Important note

- When passing arrays as functions you can do it as one of the following ways...
- `void printPassOrFail(int grades[], int size)`
or
- `void printPassOrFail(int grades[5], int size)`
- C++ allows this because during the time we write the code, we might not know how big grades array will be.
- The additional size variable will help keep track of that.

Pass by Value or Pass by Reference?

- When we pass variables into sub function, default behavior is always pass by value.
- If we need to pass by reference, we have to tell C++ with the & symbol.
- When we pass arrays into sub function, arrays are always pass by **reference**. **Sub functions are free to** modify the contents of the array.

Final Note

- A locally declared array can **NOT** be returned to the calling function.
- Example: (Don't do it!!)

```
int [] getInput();  
{  
    int grades[10];  
    //get user input;  
    return grades;  
}
```