CS 92 Competitive Programming

Silver Level

LECTURE 2: ARRAY

DR. ERIC CHOU

IEEE SENIOR MEMBER

Overview

SECTION 1

C++ Array

- C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Array

Declaring Arrays

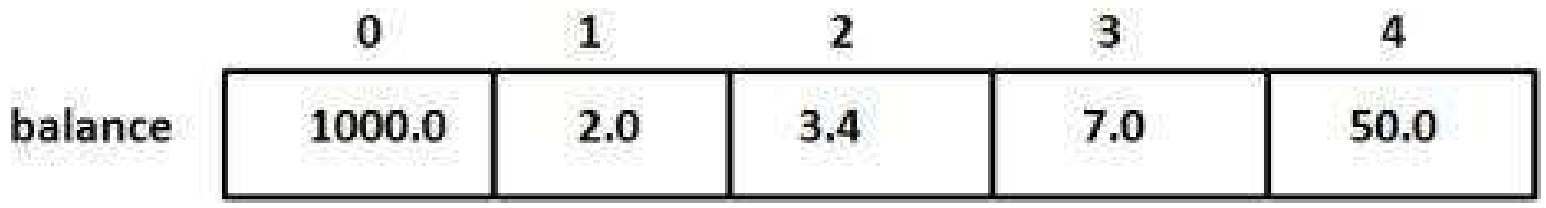
To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [arraySize];

e.g.

```
double balance[10];
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
double balance[]  = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

Array



```
int n = sizeof(arr)/sizeof(arr[0]);
```

Pointer to an Array

```
double balance[50];
```

balance is a pointer to &balance[0], which is the address of the first element of the array balance. Thus, the following program fragment assigns p the address of the first element of balance –

```
double *p;
double balance[10];
p = balance;
```

Address Calculation by Pointer

- It is legal to use array names as constant pointers, and vice versa. Therefore, *(balance + 4) is a legitimate way of accessing the data at balance[4].
- Once you store the address of first element in p, you can access array elements using *p, *(p+1), *(p+2) and so on.

Passing Arrays to Functions

Way-1

Formal parameters as a **pointer** as follows –

```
void myFunction(int *param) {
   .
   .
   .
}
```

Passing Arrays to Functions

Way-2

Formal parameters as a sized array as follows –

```
 void myFunction(int param[10]) {
      .
      .
      .
 }
```

Passing Arrays to Functions

Way-3

Formal parameters as an unsized array as follows –

```
 void myFunction(int param[]) {
      .
      .
      .
 }
```

Passing Arrays to Functions with Length

- Now, consider the following function, which will take an array as an argument along with another argument and based on the passed arguments, it will return average of the numbers passed through the array as follows –

```
double getAverage(int arr[], int size) {
   int i, sum = 0;
   double avg;
    for (i = 0; i < size; ++i) {
        sum += arr[i];
    }
    avg = double(sum) / size;

    return avg;
}
```

Return Array from Functions

- C++ does not allow to return an entire array as an argument to a function. However, you can return a pointer to an array by specifying the array's name without an index.

```
int * myFunction() {
      .
      .
      .
}
```

Return Array from Functions

- Second point to remember is that C++ does not advocate to return the address of a local variable to outside of the function so you would have to define the local variable as static variable.

```cpp
#include <iostream>
#include <ctime>
using namespace std;
// function to generate and retrun random numbers.
int *getRandom( ) {
  static int  r[10];
  // set the seed
  srand( (unsigned)time( NULL ) );

  for (int i = 0; i < 10; ++i) {
    r[i] = rand();
    cout << r[i] << endl;
  }
  return r;
}

int main () {
  int *p;
  p = getRandom();
  for ( int i = 0; i < 10; i++ ) {
    cout << "*(p + " << i << ") : ";
    cout << *(p + i) << endl;
  }
  return 0;
}
```

array_return.cpp

```
7163
13900
27101
16399
16759
27956
26189
8579
27153
6341
*(p + 0) : 7163
*(p + 1) : 13900
*(p + 2) : 27101
*(p + 3) : 16399
*(p + 4) : 16759
*(p + 5) : 27956
*(p + 6) : 26189
*(p + 7) : 8579
*(p + 8) : 27153
*(p + 9) : 6341
```

Anonymous Array

SECTION 1

Anonymous Array

- If you're using older C++ variants (pre-C++0x), then this is not allowed. The "anonymous array" you refer to is actually an initializer list. Now that C++11 is out, this can be done with the built-in initializer_list type. You theoretically can also use it as a C-style initializer list by using extern C, if your compiler parses them as C99 or later.

- **For example:**

```
int main() {
    const int* p;
    p = (const int[]){1, 2, 3};
}
```

IndexOf

find, find_if

SECTION 1

Naive solution

- Simple solution would be to write our own custom routine for finding the index of first occurrence of an element. The idea is to perform a linear search on the given array for determining the index. The approach is demonstrated below:

array_naive_find.cpp

```cpp
#include <iostream>
using namespace std;

int main(){
   int arr[] = { 6, 3, 5, 2, 8 };
   int n = sizeof(arr)/sizeof(arr[0]);
   int elem = 2;
   int i = 0;
   while (i < n){
      if (arr[i] == elem) break;
      i++;
   }

   if (i < n) {
      cout << "Element " << elem << " is present at index " << i
          << " in the given array";
   }
   else {
      cout << "Element is not present in the given array";
   }
   return 0;
}
```

```
Element 2 is present at index 3 in the given array
```

std::find algorithm

- We can also use std::find algorithm which returns an iterator that points to the target value. It defined in the <algorithm> header. To get the required index, apply pointer arithmetic or make a call to std::distance.

array_stl_find.cpp

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
   int arr[] = { 6, 3, 5, 2, 8 };
   int n = sizeof(arr)/sizeof(arr[0]);
   int elem = 2;

   auto itr = find(arr, arr + n, elem);

   if (itr != end(arr)) {
      cout << "Element " << elem << " is present at index "
          << distance(arr, itr) << " in the given array";
   }
   else {
      cout << "Element is not present in the given array";
   }
   return 0;
}
```

```
Element 2 is present at index 3 in the given array
```

std::find_if algorithm

- Sometimes it is desired to search for an element which meets certain conditions in the array. For instance, find the index of first 2-digit number in the array. To handle such cases, the recommended approach is to use the **std::find_if** algorithm which accepts a predicate.

array_stl_find_if.cpp

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
struct comp{
   int elem;
   comp(int const &i): elem(i) { }
   bool operator()(int const &i) {
      return (i == elem);
   }
};

int main(){
   int arr[] = { 6, 3, 5, 2, 8 };
   int n = sizeof(arr)/sizeof(arr[0]);
   int elem = 2;
   auto itr = find_if(arr, arr + n, comp(elem));

   if (itr != end(arr)) {
      cout << "Element " << elem << " is present at index " << distance(arr, itr) << " in the given array";
   }
   else {
      cout << "Element is not present in the given array";
   }
   return 0;
}
```

```
Element 2 is present at index 3 in the given array
```

Sorting

SECTION 1

Sorting of Integer Array

- #include <algorithm>
- Core comparison by < (operator<)

Sorting of Integer Array

Demo Program: array_sort.cpp

```
#include <bits/stdc++.h>
using namespace std;

int main(){
   int arr[] = {1, 5, 7, 9, 6, 7, 3, 4, 2, 0};
   int n = sizeof(arr)/sizeof(arr[0]);

   sort(arr, arr+n);
   cout << "Array after sorting using "
         "default sort is: \n";
   for (int i=0; i<n; ++i){
      cout << arr[i] << "";
   cout << endl;
   }
   return 0;
}
```

```
Array after sorting using default sort is:
0
1
2
3
4
5
6
7
8
9
```

Sorting of String Array

- #include <algorithm>
- #include <string>
- #include <array>
- Core comparison by < (operator<)

Sorting of String Array

Demo Program: array_stl_sort.cpp

```
#include <iostream>
#include <array>
#include <string>
#include <algorithm>
using namespace std;

int main(){
    array<string, 4> colors = {"blue", "black", "red", "green"};
    for (string color: colors){ cout << color << " ";  }
    cout << endl;
    sort(colors.begin(), colors.end());
    for (string color: colors){ cout << color << " ";  }
    cout << endl;
    return 0;
}
```

```
blue black red green
black blue green red
```

Sorting of struct Array

- #include <algorithm>
- Core comparison by < (operator<)

array_struct_sort.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
template <typename T> string str(const T& n){
 ostringstream stm; stm << n; return stm.str() ;
}

struct data{
    int num1;
    int num2;
    bool operator<(const data& other) const { return num1 < other.num1; }
    string to_string(){ return "("+str(num1)+", "+str(num2)+")"; }
};

int main(){
   int N=5;
   data array[N] = {{3, 5}, {5, 2}, {4, 1}, {2, 3}, {1, 4}};
   cout << "{";
   for (int i=0; i<N; i++){ cout << array[i].to_string() << " "; }
   cout << "}" << endl;
   sort(array, array+N);
   cout << "{";
   for (int i=0; i<N; i++){ cout << array[i].to_string() << " "; }
   cout << "}" << endl;
}
```

```
{(3, 5) (5, 2) (4, 1) (2, 3) (1, 4) }
{(1, 4) (2, 3) (3, 5) (4, 1) (5, 2) }
```

array_struct_sort2.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
template <typename T> string str(const T& n){
 ostringstream stm; stm << n; return stm.str() ;
}

struct data{
    int num1;
    int num2;
    string to_string(){ return "("+str(num1)+", "+str(num2)+")"; }
};
bool compare(data lhs, data rhs) { return lhs.num2 < rhs.num2; }

int main(){
   int N=5;
   data array[N] = {{3, 5}, {5, 2}, {4, 1}, {2, 3}, {1, 4}};
   cout << "{";
   for (int i=0; i<N; i++){ cout << array[i].to_string() << " "; }
   cout << "}" << endl;
   sort(array, array+N, compare);
   cout << "{";
   for (int i=0; i<N; i++){ cout << array[i].to_string() << " "; }
   cout << "}" << endl;
}
```

```
{(3, 5) (5, 2) (4, 1) (2, 3) (1, 4) }
{(4, 1) (5, 2) (2, 3) (1, 4) (3, 5) }
```