7.1: Introduction to arrays

Declaring and Referencing Arrays

An **array** behaves like a list of variables that can be declared in a single line as follows:

```
int score[n]; // Creates an array of size n
```

▼ where

- `n` is the *size* of the array.
- `score[0]`, `score[1]`, ..., `score[n-1]` are the *indexed variables* with `0`, `1`, ..., `n-1` being the *indexes*.
- `int` is the *base type* of the array.

PROGRAMMING TIP: Use a `const` for size of array

All compilers will not allow the use of *variable* for defining the size of an array hence we should always use *constants*.

```
const int SIZE = 7; // Constant defined
int score[SIZE]; // Creates an array
```

Arrays in Memory

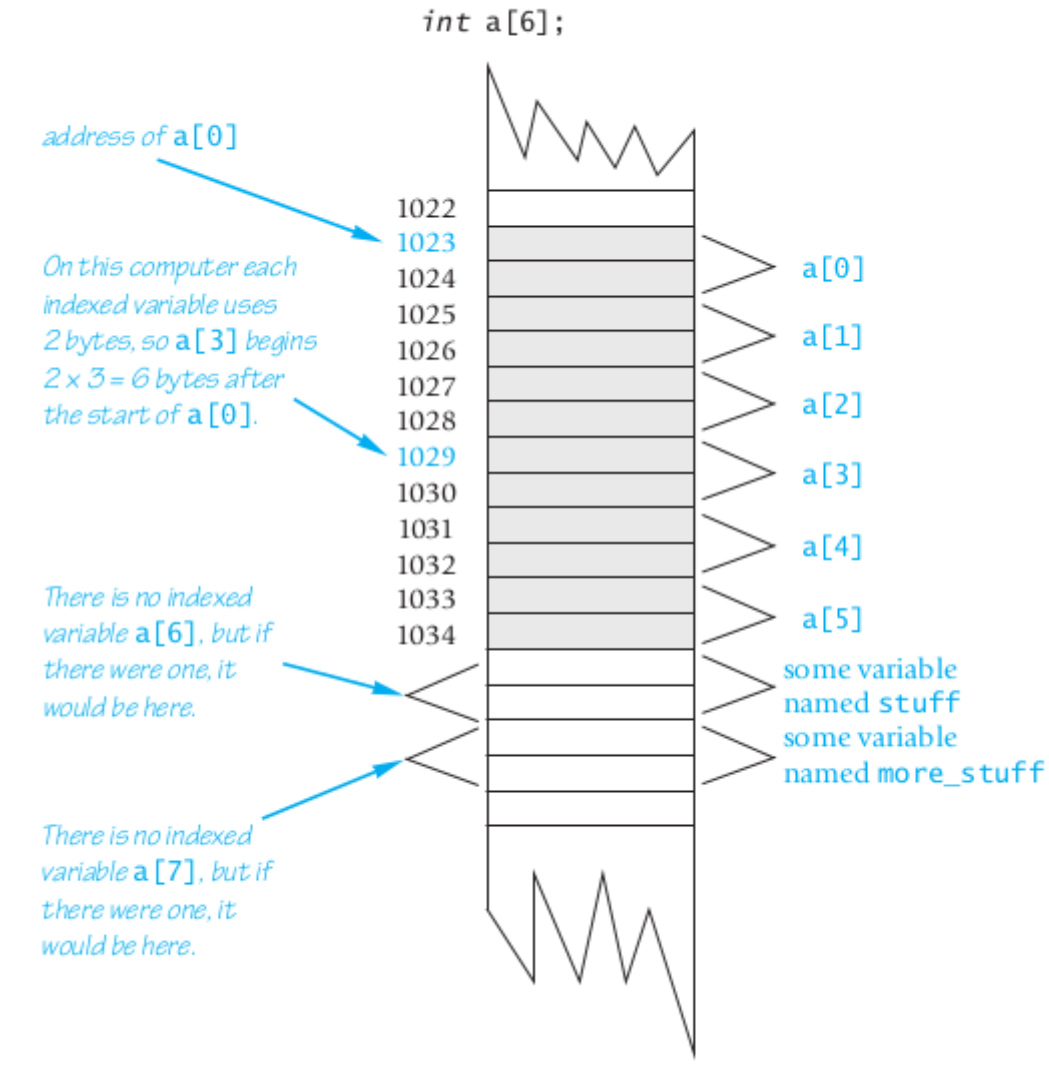
Before discussing arrays, lets see how variables are represented in the computer memory. The computer's memory consists of a list of numbered locations called *bytes*. The number of a byte is called it's *address*.

▼ Thus, a simple variable in memory is described by two pieces of information:

- An Address in memory (location of 1st byte for that variable).
- The type of variable (telling how many bytes required which will then be stored consequtively).

Array indexed variables are represented in memory the same way as ordinary variables but these indexed variables are placed next to each other in memory. For example consider, when the an array is declared `int a[6];` the computer reserves memory for six `int` variables and always places them one after another in memory. This way, it just remembers the address of `a[0]` and can get all the rest from this address as shown in figure below.

DISPLAY 7.2 An Array in Memory



PITFALL: Array Index out of range

The most common programming error when using arrays is attempting to reference a nonexistent array index. When something like this happens, it's called *out of range* or *illegal*, but the biggest problem is that the program will run **WITHOUT GIVING ANY WARNING**. What it'll do here is simply pick the value in that address location, which will destroy the whole program.

Initializing Arrays

An Array can be initialized when it is declared. For example:

```
int children[3] = {1, 2, 3};
```

or

```
int children[] = {1, 2, 3};
```

is equivalent to

```
int children[3];
children[0] = 1;
children[1] = 2;
children[2] = 3;
```

If we list fewer values than the indexed variables then those are used to initialize the first few and rest are set to 0. For example:

```
int arr[3] = {1};
```

will have `arr[0] = 1`, `arr[1] = 0` and `arr[2] = 0`.

PROGRAMMING TIP: C++11 Range_based `for` Statement

This simplifies iteration over every element in an array. For example:

```
int arr[] = {2, 4, 3, 1};
for (int x: arr)
{
  std::cout << x;
}
```

will result in

```
2431
```

The above example was equivalent to pass-by-value (change in `x` will not change the values in the array). We can use pass-by-reference and change the values in the array as we go (to avoid this mistake, we should declare arrays as constants when necessary):

```
int arr[] = {2, 4, 3, 1};
for (int& x: arr)
{
  x++;
}

for (auto x: arr)
{
  std::cout << x;
}
```

will result in

```
3542
```

7.2: Arrays in function

Both indexed variables and entire arrays can be used as arguments to functions.

Indexed Variables as Function Arguments

An indexed variable can be an argument to a function exactly like any simple varibale would.

```
// Function definition
int adjust(int oldDays)
{
  return (oldDays + 5);
}

// Array Declaration
int vacation[10];

// Function Calling
adjust(vacation[num]);
```

Entire Arrays as Function Arguments

A function can have a formal parameter for an entire array, however when it is called, it's neither a call-by-value parameter nor a call-by-reference parameter. It's a new kind called *array parameter* (but acts like a call-by-reference).

▼ An array has three parts:

- The address (location in memory) of 1st indexed variable.
- The base type
- The size

When an array is used as a formal parameter, only *address* and *base type* is given to the function which makes it different from call-by-reference.

Note that in function declaration and definition, we don't need to put the size inside `[]` of the array and during the call, we don't even use `[]` in arguments.

▼ For Example:

- **FUNCTION DECLARATION**

```
void fill_up(int a[], int size);
//Precondition: size is the declared size of the array a.
//The user will type in size integers.
//Postcondition: The array a is filled with size integers
//from the keyboard.
```

- **FUNCTION DEFINITION**

```
void fill_up(int a[], int size)
{
  std::cout << "Enter " << size << " numbers:\\n";
  for (int i = 0; i < size; i++)
    std::cin >> a[i];
  size--;
  std::cout << "The last array index used is " << size << endl;
}
```

- **FUNCTION CALL**

```
int score[5], numScores = 5;
fill_up(score, numScores);
```

After the function is run, the array `score` will contain the values `[0, 1, 2, 3, 4]`.

The `const` parameter modifier

As we've seen that *array parameter* can change the value of indexed variable, hence if we don't want to do this, it's advised to use `const` in formal parameters. For example:

```
void show(const int a[], int size)
{
  // BODY OF THE FUNCTION
}
```

Now if the function tries to change the value, there will be an error or warning.

PITFALL: Inconsistent use of `const` parameters

If we use `const` for one array parameter of paarticular type, then should be used for every other array parameter of that type. This is maily due to the function calls within function. For example, in code below

```
double compute_average(int a[ ], int number_used);
//Returns the average of the elements in the first number_used
//elements of the array a. The array a is unchanged.
void show_difference(const int a[ ], int number_used)
{
  double average = compute_average(a, number_used);
  std::cout << "Average of the " << number_used <<
    " numbers = " << average << endl
    << "The numbers are:\\n";
  for (int index = 0; index < number_used; index++)
    std::cout << a[index] << " differs from average by "
      << (a[index] - average) << endl;
}
```

will generate an error or warning because we are saying don't modify `a` in `show_difference` but no such information is conveyed when calling `compute_average`. To rectify this, just modify formal parameter of `compute_average` to:

```
double compute_average(const int a[ ], int number_used);
```

7.3: Programming with Arrays

▼ Things discussed here:

💡 Partially filled Arrays

💡 Sorting and searching in arrays

Partially Filled Arrays

Partially filled arrays require some care. The program must keep track of how much of the array is used and must not reference any indexed variable that has not been given a value.

Programming Tip: Do not Skimp on formal parameters

When defining functions, its good habit to use global variables as formal parameters. For example

```
fillArray(score, MAX_SCORES, num_used);
```

PROGRAMMING EXAMPLE: Searching an Array

▼ Code

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>

int ARR_SIZE = 20;

// Function to declare array
void arrDec(int arr[], int ARR_SIZE, int listSize, int maxInt);

// Function to search array
void arrSearch(const int arr[], int listSize, int num);

// Function to output array
void output(const int arr[], int listSize);

int main()
```

```
{
    srand(time(0));

    int arr[ARR_SIZE], listSize, maxInt;
    std::cout << "Enter the number of elements you want in array (max 20): ";
    std::cin >> listSize;

    std::cout << "Enter the max number you want in array: ";
    std::cin >> maxInt;

    arrDec(arr, ARR_SIZE, listSize, maxInt);

    int num;
    std::cout << "Enter the number you want to search: ";
    std::cin >> num;

    arrSearch(arr, listSize, num);

    std::cout << "Array: ";
    output(arr, listSize);
    return 0;
}

void arrDec(int arr[], int ARR_SIZE, int listSize, int maxInt)
{
    if (listSize > ARR_SIZE)
    {
        std::cout << "Max size allowed for array is "
        << ARR_SIZE << std::endl;
        return;
    }

    for (int i = 0; i < listSize; i++)
        arr[i] = (rand() % maxInt);
}

void arrSearch(const int arr[], int listSize, int num)
{
    for (int i = 0; i < listSize; i++)
    {
        if (arr[i] == num)
        {
            std::cout << "Number present \n";
            return;
        }
    }
    std::cout << "Number not present \n";
}

void output(const int arr[], int listSize)
{
    for (int i = 0; i < listSize; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
}
```

PROGRAMMING EXAMPLE: Selection Sort

▼ Code

```cpp
#include <iostream>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>

int ARR_SIZE = 20;

// Function to declare array
void arrDec(int arr[], int ARR_SIZE, int listSize, int maxInt);

// Function to find the index of smallest element in array
int smallestIndex(const int arr[], int start, int end);

// Function to swap two elements of array
void swap(int& e1, int&e2);

// Function to sort
void sort(int arr[], int listSize);

// Function to output array
void output(const int arr[], int listSize);

int main()
{
    srand(time(0));

    int arr[ARR_SIZE], listSize, maxInt;
    std::cout << "Enter the number of elements you want in array (max 20): ";
    std::cin >> listSize;

    std::cout << "Enter the max number you want in array: ";
    std::cin >> maxInt;

    arrDec(arr, ARR_SIZE, listSize, maxInt);
    std::cout << "Initialized Array: ";
    output(arr, listSize);

    sort(arr, listSize);
    std::cout << "Sorted Array: ";
    output(arr, listSize);
}

void arrDec(int arr[], int ARR_SIZE, int listSize, int maxInt)
{
    if (listSize > ARR_SIZE)
    {
        std::cout << "Max size allowed for array is "
        << ARR_SIZE << std::endl;
        return;
    }

    for (int i = 0; i < listSize; i++)
        arr[i] = (rand() % maxInt);
}

void sort(int arr[], int listSize)
{
    for (int i = 0; i < listSize-1; i++)
    {
        int idxSmallest = smallestIndex(arr, i, listSize);
```

```
        swap(arr[i], arr[idxSmallest]);
    }
}

int smallestIndex(const int arr[], int start, int end)
{
    int idxSmallest = start;
    for (int j = start+1; j < end; j++)
    {
        if (arr[j] < arr[idxSmallest])
            idxSmallest = j;
    }
    return idxSmallest;
}

void swap(int& e1, int&e2)
{
    int temp = e1;
    e1 = e2;
    e2 = temp;
}

void output(const int arr[], int listSize)
{
    for (int i = 0; i < listSize; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
}
```

PROGRAMMING EXAMPLE: Bubble Sort

▼ Code

```
#include <iostream>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>

int ARR_SIZE = 20;

// Function to declare array
void arrDec(int arr[], int ARR_SIZE, int listSize, int maxInt);

// Function to swap two elements of array
void swap(int& e1, int&e2);

// Function to sort
void sort(int arr[], int listSize);

// Function to output array
void output(const int arr[], int listSize);

int main()
{
    srand(time(0));

    int arr[ARR_SIZE], listSize, maxInt;
```

```
    std::cout << "Enter the number of elements you want in array (max 20): ";
    std::cin >> listSize;

    std::cout << "Enter the max number you want in array: ";
    std::cin >> maxInt;

    arrDec(arr, ARR_SIZE, listSize, maxInt);
    std::cout << "Initialized Array: ";
    output(arr, listSize);

    sort(arr, listSize);
    std::cout << "Sorted Array: ";
    output(arr, listSize);
}

void arrDec(int arr[], int ARR_SIZE, int listSize, int maxInt)
{
    if (listSize > ARR_SIZE)
    {
        std::cout << "Max size allowed for array is "
        << ARR_SIZE << std::endl;
        return;
    }

    for (int i = 0; i < listSize; i++)
        arr[i] = (rand() % maxInt);
}

void sort(int arr[], int listSize)
{
    for (int i = listSize-1; i > 0; i--)
    {
        for (int j = 0; j < i; j++)
        {
            if (arr[j] > arr[j+1])
            {
                swap(arr[j], arr[j+1]);
            }
        }
    }
}

void swap(int& e1, int&e2)
{
    int temp = e1;
    e1 = e2;
    e2 = temp;
}

void output(const int arr[], int listSize)
{
    for (int i = 0; i < listSize; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
}
```

EXAMPLE: String Sort

▼ Code

```cpp
#include <iostream>
#include <cstring>

int MAX_SIZE = 20;

// Function to input word
void input(char word[], int MAX_SIZE, int& sizeUsed);

// Function to swap two elements of array
void swap(char& e1, char&e2);

// Function to sort
void sort(char arr[], int listSize);

// Function to output word
void output(const char word[], int sizeUsed);

int main()
{
    int sizeUsed;
    char word[MAX_SIZE];
    input(word, MAX_SIZE, sizeUsed);
    sort(word, sizeUsed);
    std::cout << "Sorted word: ";
    output(word, sizeUsed);
}

void input(char word[], int MAX_SIZE, int& sizeUsed)
{
    std::cout << "Write the characters of the word (max 20) and end by hitting enter: ";
    char letter;
    int idx = 0;
    do
    {
        std::cin.get(letter);
        word[idx] = letter;
        idx++;
    } while (letter != '\n' && idx < MAX_SIZE+1);
    sizeUsed = idx;
}

void sort(char arr[], int listSize)
{
    for (int i = listSize-1; i > 0; i--)
    {
        for (int j = 0; j < i; j++)
        {
            if (arr[j] > arr[j+1])
            {
                swap(arr[j], arr[j+1]);
            }
        }
    }
}

void swap(char& e1, char&e2)
{
    int temp = e1;
    e1 = e2;
    e2 = temp;
}
```

```
void output(const char word[], int sizeUsed)
{
    for (int i = 1; i < sizeUsed; i++)
        std::cout << word[i];
    std::cout << std::endl;
}
```

7.4: Multidimensional Array

The following declares an array called `page` which has two indixes `char page[30][100];`.

A 2D array can be visualized as the 2D display with 1st index giving the row and 2nd index giving the column.

```
page[0][0], page[0][1], ..., page[0][99]
page[1][0], page[1][1], ..., page[1][99]
page[2][0], page[2][1], ..., page[2][99]
.
.
.
page[29][0], page[29][1], ..., page[29][99]
```

In C++, a 2D array is actually an array of arrays. The example `page` is actually a 1D array of size 30 whose base type is a 1D array of charactuers of size 100.

Multidimensional Array Parameters

Viewing 2D array as an array of arrays will help understand how C++ handles parameters for multidimensional arrays.

```
void display_page(const char p[][100], int size_dimension_1)
{
    for (int index1 = 0; index1 < size_dimension_1; index1++)
    {//Printing one line:
        for (int index2 = 0; index2 < 100; index2++)
            cout << p[index1][index2];
        cout << endl;
    }
}
```

In the function's formal paramters, the first dimension is really the index of the array and is treated just like an array index for an ordinary, 1D array. The second dimension is part of the description of the base type, which is an array of characters of size 100.

PROGRAMMING EXAMPLE: 2D Grading Program

▼ Code

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>

const int NUM_STUDENTS = 4, NUM_QUIZES = 3;

// Function to declare arrays
void arrDec(int arr[][NUM_QUIZES], int numRows = NUM_STUDENTS,
            int numCols = NUM_QUIZES, int maxInt = 10);
```

```cpp
// Function to view a 2D array
void view(const int arr[][NUM_QUIZES], int NUM_STUDENTS = NUM_STUDENTS);

// Function to calculate student average
void calcStAvg(const int arr[][NUM_QUIZES], double studentAvg[],
               int NUM_STUDENTS = NUM_STUDENTS);

// Function to calculate quiz average
void calcQzAvg(const int arr[][NUM_QUIZES], double quizAvg[],
               int NUM_STUDENTS = NUM_STUDENTS);


// Function to display output
void output(const int arr[][NUM_QUIZES], const double quizAvg[],
            const double studentAvg[]);

int main()
{
    srand(time(0));

    int arr[NUM_STUDENTS][NUM_QUIZES];
    double studentAvg[NUM_STUDENTS], quizAvg[NUM_QUIZES];

    arrDec(arr);
    calcStAvg(arr, studentAvg);
    calcQzAvg(arr, quizAvg);

    output(arr, quizAvg, studentAvg);
    return 0;
}

void arrDec(int arr[][NUM_QUIZES], int numRows, int numCols, int maxInt)
{
    for (int i = 0; i < numRows; i++)
    {
        for (int j = 0; j < NUM_QUIZES; j++)
        {
            arr[i][j] = (rand() % maxInt);
        }
    }
}

void calcStAvg(const int arr[][NUM_QUIZES], double studentAvg[], int NUM_STUDENTS)
{
    for (int i = 0; i < NUM_STUDENTS; i++)
    {
        double count = 0;
        for (int j = 0; j < NUM_QUIZES; j++)
        {
            count += arr[i][j];
        }
        studentAvg[i] = count / static_cast<double>(NUM_QUIZES);
    }
}

void calcQzAvg(const int arr[][NUM_QUIZES], double quizAvg[], int NUM_STUDENTS)
{
    for (int i = 0; i < NUM_QUIZES; i++)
    {
        double count = 0;
        for (int j = 0; j < NUM_STUDENTS; j++)
        {
            count += arr[j][i];
        }
        quizAvg[i] = count / static_cast<double>(NUM_STUDENTS);
    }
}
```

```
void output(const int arr[][NUM_QUIZES], const double quizAvg[], const double studentAvg[])
{
    std::cout.setf(std::ios::fixed);
    std::cout.setf(std::ios::right);
    std::cout.setf(std::ios::showpoint);
    std::cout.precision(1);

    std::cout << std::setw(10) << "Student"
            << std::setw(5) << " Ave"
            <<std::setw(15) << "Quizes" << std::endl;
    for (int i = 0; i < NUM_STUDENTS; i++)
    {
        std::cout << std::setw(10) << i+1
                << std::setw(5) << studentAvg[i];
        for (int j = 0; j < NUM_QUIZES; j++)
        {
            std::cout << std::setw(5) << arr[i][j];
        }
        std::cout << std::endl;
    }
    std::cout << "Quiz Averages: ";
    for (int k = 0; k < NUM_QUIZES; k++)
    {
        std::cout << std::setw(5) << quizAvg[k];
    }
    std::cout << std::endl;
}

void view(const int arr[][NUM_QUIZES], int NUM_STUDENTS)
{
    for (int i = 0; i < NUM_STUDENTS; i++)
    {
        for (int j = 0; j < NUM_QUIZES; j++)
        {
            std::cout << std::setw(5) << arr[i][j];
        }
        std::cout << std::endl;
    }

}
```

EXAMPLE: String Arrays

▼ Code

```
/*
Author:         Tushar Gautam
Date:           23rd Jan 2021
Description:    -> Find phone number corresponding to
                the name

                Inputs:
                1) 1D Array of names
                2) 1D Array of phone numbers

                Outputs:
                1) The number
*/

#include <iostream>
#include <cstring>
```

```
// Function to look up string
std::string lookupName(std::string name, std::string names[],
                       std::string numbers[], int size);

int main()
{
    std::string names[] = {"Michael Myers",
                   "Ash Williams",
                   "Jack Torrance",
                   "Freddy Krueger"};
    std::string phoneNumbers[] = {"333-8000","333-2323",
                          "333-6150","339-7970"};

    std::string targetName, targetPhone;
    char c;
    do
    {
        std::cout << "Enter a name to find the "
            << "corresponding phone number."
            << std::endl;
        std::getline(std::cin, targetName);
        targetPhone = lookupName(targetName,
        names, phoneNumbers,4);

        if (targetPhone.length() > 0)
            std::cout << "The number is: " << targetPhone << std::endl;
        else
            std::cout << "Name not found. " << std::endl;

        std::cout << "Look up another name? (y/n)"
            << std::endl;
        std::cin >> c;
        std::cin.ignore();
    } while (c == 'y');
    return 0;
}

std::string lookupName(std::string name, std::string names[], std::string numbers[], int size)
{
    for (int i = 0; i < size; i++)
    {
        if (names[i] == name)
        {
            return numbers[i];
        }
    }
    return " Not in directory";
}
```