Introduction to Object-Oriented Programming

COMP2011: Array — a Collection of Homogeneous Objects

Dr. Cindy Li
Dr. Brian Mak
Dr. Dit-Yan Yeung

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China

What is an Array?

- Array is a collection of homogeneous objects: objects of the same type. e.g. a collection of int, char, double, ..., or user-defined types.
- Exception: The array elements cannot be reference variables.

Motivation from Programming Point of View

- A function to sort 3 integers can be:

```
void sort_3_int(int& x, int& y, int& z);
```

- A function to sort 6 integers can be:

```
void sort_6_int(int& u, int& v, int& w, int& x, int& y, int& z);
```

- How about a function to sort 10,000 integers? Are you going to create variable names for the 10,000 different integers?
- Array is designed to solve this problem: you only need one identifier name to address all the 10,000 integers, and there is a way to refer to each of them.
- It can solve problems like: read a list of student names, and sort them in alphabetical order.
- In an Excel file, each column/row is basically an array so that you can do some common operations (like average, max, min, count) on it.

Part I

1-Dimensional Array

C++ 1-Dimensional Array

Syntax: Definition of a 1D Array

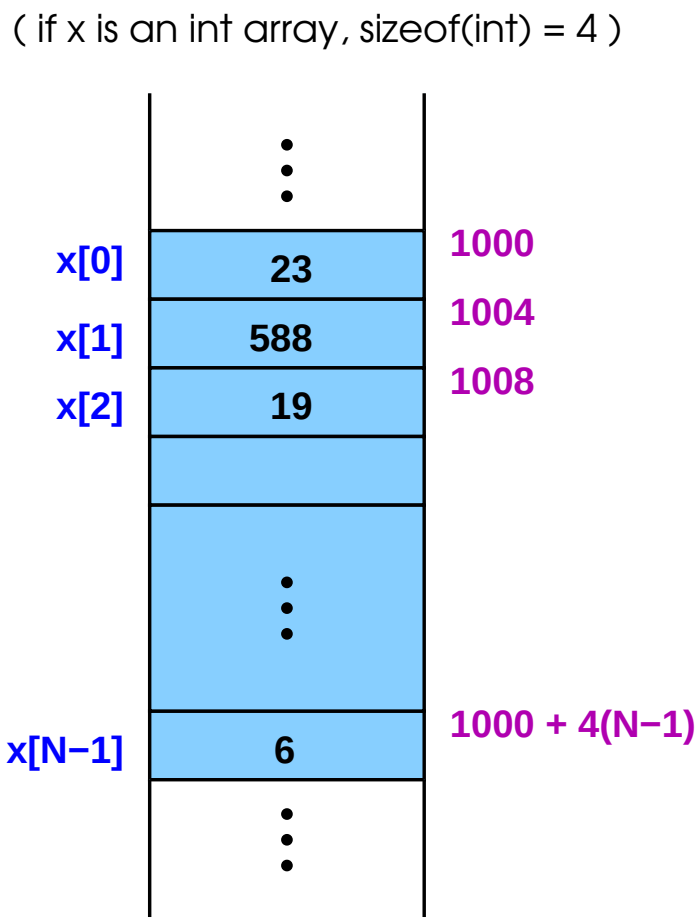
<data-type> <array-name> [<size>] ;

- <size> should be a positive constant. It can be a constant expression too.

Examples

```
int number[10000];                  // an array of 10,000 uninitialized integers

const int NUM_STUDENTS = 335;
char gender[NUM_STUDENTS];                      // an array of 335 char
float score[NUM_STUDENTS + 1]; // an extra element to hold the mean

int n = 3;
double x[n];          // compilation error on VC++: size is NOT a constant

int value[-4];                    // compilation error: array size cannot be -ve
```

Subscripting: Access to Each Array Element



- A 1D array is an ordered list of elements.
- Successive elements are stored in contiguous memory.
- To access an element, use the subscript operator [] with an array index.
- For an array of size N, the indices run from $0, 1, 2, \ldots, N-1$.
- Each array element is treated like a regular variable:
  - you may assign a value to it
  - you may assign its value to another variable
  - you may pass it by value or reference to a function

Array and Control

- Array works particularly well with loops: e.g. use a for-loop to access and manipulate each array element in turn.
- This is not a coincidence, but part of the C++ language design.

Examples

```
int y;                                          // A regular int variable
int x[3];                                       // An array of 3 int numbers

x[0] = 34;                                      // Array indices start from zero in C++
x[1] = 289;
x[2] = 75;                                      // Index of the last element is 2 NOT 3!

y = x[2];                                       // Now both y and x[2] are 75
max(x[2], x[0]);                                // Pass array elements by value
swap(x[1], x[0]);                               // Pass array elements by reference

for (int j = 0; j < 3; j++)                     // Triple each element of an array
    x[j] *= 3;
```

Example: Manipulate an Array of Scores using for Loop

```
#include <iostream>                                    /* array-mean.cpp */
using namespace std;

int main(void)
{
    const int NUM_STUDENTS = 5;
    float score[NUM_STUDENTS];
    // Read in the first student's score. Assume #student >= 1
    cin >> score[0];
    // Don't forget initializing the sum of scores
    float sum_score = score[0];

    for (int j = 1; j < NUM_STUDENTS; ++j)
    {
        cin >> score[j];
        sum_score += score[j];                    // Accumulate the scores
    }

    cout << "mean score = " << sum_score/NUM_STUDENTS << endl;
    return 0;
}
```

Example: Manipulate an Array of Scores using for Loop ..

```
#include <iostream>                                    /* array-max.cpp */
using namespace std;

int main(void)
{
    const int NUM_STUDENTS = 5;
    float score[NUM_STUDENTS];
    // Read in the first student's score. Assume #student >= 1
    cin >> score[0];
    float max_score = score[0];   // Don't forget initializing the max score

    for (int j = 1; j < NUM_STUDENTS; ++j)
    {
        cin >> score[j];
        if (max_score < score[j])
            max_score = score[j];
    }

    cout << "max score = " << max_score << endl;
    return 0;
}
```

Wrong Subscript: Common Reason for Segmentation Fault

- C++ compiler does not automatically check that an array index is out of bound.
- That is, for an array of size N, the compiler won't check if it is subscripted with an index between 0 and $N-1$, neither at compile-time nor run-time.
- There is no compilation error for the following codes:

```
int x[10]; x[-2] = 5; x[100] = 9;
```

- When the codes are run, `x[-2] = 5;` will put the value 5 to the memory space which is 2×4 bytes (size of 2 int) before the array x. Similarly, `x[100] = 9;` will put the value 9 to the memory space which is 90×4 bytes beyond the array.
- This is a common cause of the run-time error called segmentation fault — your program trespasses into memory locations that do not belong to it.

Array Initialization

- Just like any local variable, when an array is defined, its elements are not initialized automatically.

Syntax: Define and Initialize a 1D Array Simultaneously

<data-type> <array-name> [<size>]
= { <value$_0$>, <value$_1$>, . . . , <value$_{<size>-1}$> } ;

- If there are fewer values than the array size, the unspecified values will be zeros.
- It is a compilation error if there are more values than the array size.
- If you leave out the array size in the array initialization, the compiler will count the number of initializing values and uses that as the array size.
- Once defined, you cannot assign values to an array using the initialization syntax.

Example: Array Initialization

```
int a[5] = {1, 2, 3, 4, 5};
/* Same as

    int a[5];
    a[0] = 1; a[1] = 2; a[2] = 3; a[3] = 4; a[4] = 5;
*/

int b[5] = {1, 2};                    // => {1, 2, 0, 0, 0}
int c[5] = {};                        // => {0, 0, 0, 0, 0}

int d[] = {1, 2, 3}; // Compiler will determine the size automatically as 3

int e[3];
e = {5, 6, 7}; // Compilation error: can't assign values to an array like this

// Compilation error: can't declare an array of references
double x = 1.5, y = 2.5, z = 3.5;
int& s[] = {x, y, z};
```

Common Mis-uses of an Array

While each array element can be treated as a simple variable, the whole array, as represented by the array identifier, cannot.

Examples: Correct and Incorrect Uses of Arrays

```
int x[ ] = {1, 2, 3, 4, 5 };
int y[ ] = {6, 7, 8, 9, 0 };
int z[5];

/* Incorrect way */
x = {5, 4, 3, 2, 1};                    // Cannot assign to each array element
                                        // using the initialization syntax
x = 8;                                  // x is not an integer! Its elements are.
x += 2;                                 // x is not an integer! Its elements are.
x = y;                                  // No assignment between 2 arrays
z = x + y;          // Cannot +, -, *, / on the array name, but only its elements

/* Correct way; what does each for-statement do? */
for (int j = 0; j < 5; ++j) x[j] = 5 - j;
for (int j = 0; j < 5; ++j) x[j] = 8;
for (int j = 0; j < 5; ++j) x[j] += 2;
for (int j = 0; j < 5; ++j) x[j] = y[j];
for (int j = 0; j < 5; ++j) z[j] = x[j] + y[j];
```

Pass a 1D Array to a Function

Examples: Arrays as Function Arguments

```
/* function header */
float mean_score(float score[ ], int size) { ... }
float max_score(float score[ ], int size) { ... }


/* inside the main() */
float score[NUM_STUDENTS];
mean_score(score, NUM_STUDENTS);
max_score(score, NUM_STUDENTS);
```

- Since the array identifier alone does *not* tell us about its size, a function that operates on an array needs at least 2 input arguments:
  - the array identifier
  - the array size (of type int)

Example: Pass an Array to a Function

```
#include <iostream>                                                    /* array-mean-max-fcn.cpp */
using namespace std;

float mean_score(float score[ ], int size)
{
    float sum_score = 0.0;                                  // don't forget initializing the sum to zero
    for (int j = 0; j < size; j++)
        sum_score += score[j];                                          // accumulate the scores
    return sum_score/size;
}

float max_score(float score[ ], int size)
{
    float max_score = score[0];                          // initialize the max score to that of the first student
    for (int j = 1; j < size; j++)
        if (max_score < score[j])
            max_score = score[j];
    return max_score;
}

int main(void)
{
    const int NUM_STUDENTS = 5;
    float score[NUM_STUDENTS];

    for (int j = 0; j < NUM_STUDENTS; j++)
        if (!(cin >> score[j])) return -1;

    cout << "mean score = " << mean_score(score, NUM_STUDENTS) << endl;
    cout << "max score = " << max_score(score, NUM_STUDENTS) << endl;
    return 0;
}
```

1D Array as a Function's Formal Parameter

- While a regular variable may be passed to a function by value or reference, an array variable is always passed by value.
- However, although the array variable is passed by value, its elements are *effectively* passed by reference!
- Any change to an array element inside the function will persist even after the function returns.
- Just like a regular variable, you pass an array to a function simply by its variable name. e.g.

```
max_score(score, NUM_STUDENTS);
```

Example: Modifying Array's Elements by a Function

```
#include <iostream>                                                    /* array-add-rotate.cpp */
using namespace std;

void array_add(int x[ ], int y[ ], int z[ ], int size)
{
    for (int j = 0; j < size; j++)
        z[j] = x[j] + y[j];
}

void circular_rotation(int x[ ], int size)
{
    int item_0 = x[0];                                  // save the first element before rotation
    for (int j = 1; j < size; j++)
        x[j-1] = x[j];                                                 // rotate up
    x[size - 1] = item_0;                                          // fix the last element
}

void array_print(int x[ ], int size)
{
    for (int j = 0; j < size; j++)
        cout << x[j] << '\t';
    cout << endl;
}

int main(void)
{
    int a[ ] = {1, 2, 3, 4}; int b[ ] = {11, 12, 13, 14}; int c[4];
    array_add(a, b, c, 4); array_print(c, 4); cout << endl;

    for (int k = 0; k < 4; k++) { circular_rotation(a, 4); array_print(a, 4); }
    return 0;
}
```

Constant Array

- Just like simple constants, an array of constants can be made using the keyword "const".

```
const int x[] = { 1, 2, 3, 4 };
```

  It defines 4 integer constants: x[0], x[1], x[2], and x[3] are all of the type const int.

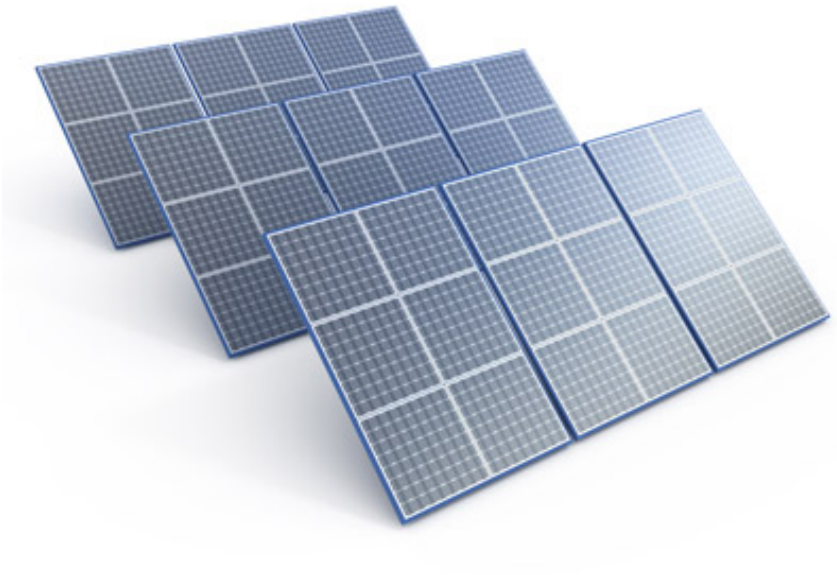
- Like simple constants, a constant array
  - must be initialized when it is defined.
  - once defined, its elements cannot be modified.
- One main use of constant array is in the definition of the formal parameters of a function: to disallow modification of the elements of an array passed to a function, declare that array constant using const.
  - inside the function, the array is read-only.
  - however, the original array in the caller is still writable.

Example: Prevent Modification by Constant Array

```
#include <iostream>                                          /* const-array-mean-max-fcn.cpp */
using namespace std;

float mean_score(const float score[ ], int size)
{
    float sum_score = 0.0;                                   // don't forget initializing the sum to zero
    for (int j = 0; j < size; j++)
        sum_score += score[j];                               // accumulate the scores
    return sum_score/size;
}

float max_score(const float score[ ], int size)
{
    float max_score = score[0];                              // initialize the max score to that of the first student
    for (int j = 1; j < size; j++)
        if (max_score < score[j])
            max_score = score[j];
    return max_score;
}

int main(void)
{
    const int NUM_STUDENTS = 5;
    float score[NUM_STUDENTS];

    for (int j = 0; j < NUM_STUDENTS; j++)
        if (!(cin >> score[j])) return -1;

    cout << "mean score = " << mean_score(score, NUM_STUDENTS) << endl;
    cout << "max score = " << max_score(score, NUM_STUDENTS) << endl;
    return 0;
}
```

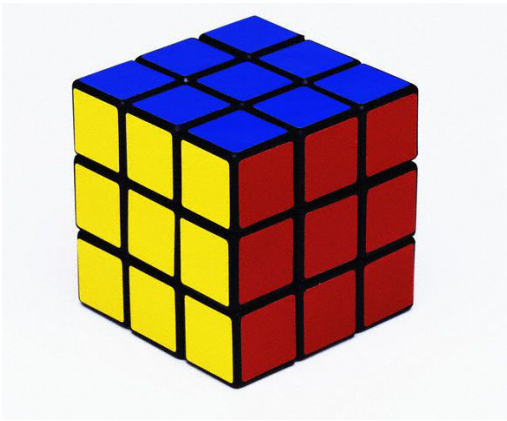
Part II

Multi-dimensional Array



Array of any Dimensions

In general, an array can be multi-dimensional.

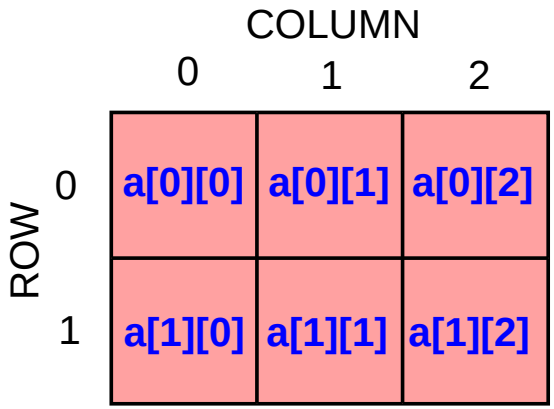


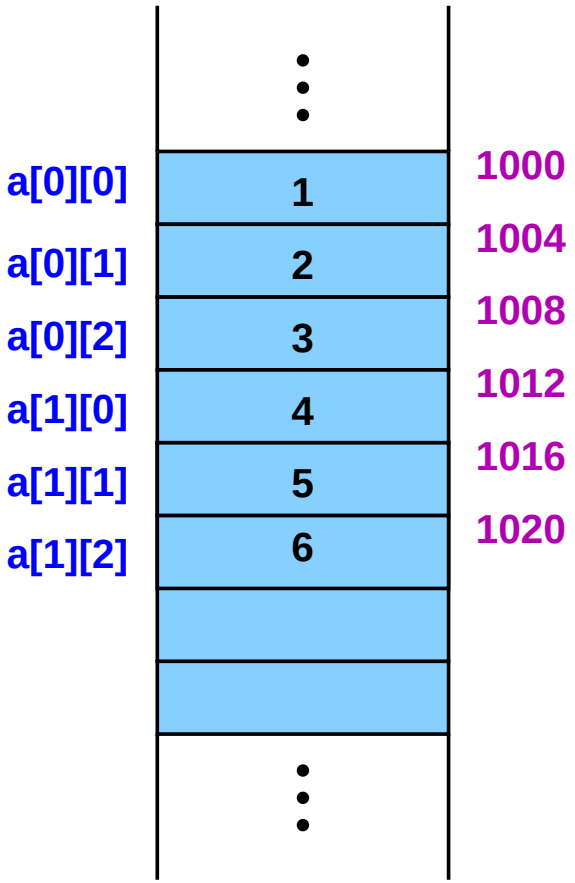
C++ 2-dimensional Array

Syntax: Definition of a 2D Array

<data-type> <array-name> [<size$_1$>] [<size$_2$>] ;

```
int a[2][3] = {1,2,3,4,5,6}; // sizeof(int) = 4
```





Initialization of 2D Array

- A 2D array can be initialized in 2 ways:
  - row by row, or
  - like a 1D array since the array cells are actually stored linearly in the memory.

Examples

```
/* Initialize row by row */
int point[5][2] = {                    // an int array with 5 rows and 2 columns
     {1, 1},
     {2, 4},
     {3, 9},
     {4, 16},
     {5, 25}
};

/* Initialize using the fact that the cells of a 2D
   array actually are stored linearly in the memory
*/
int point[5][2] = { 1,1, 2,4, 3,9, 4,16, 5,25 };
```

Example: Functions with 2D Array

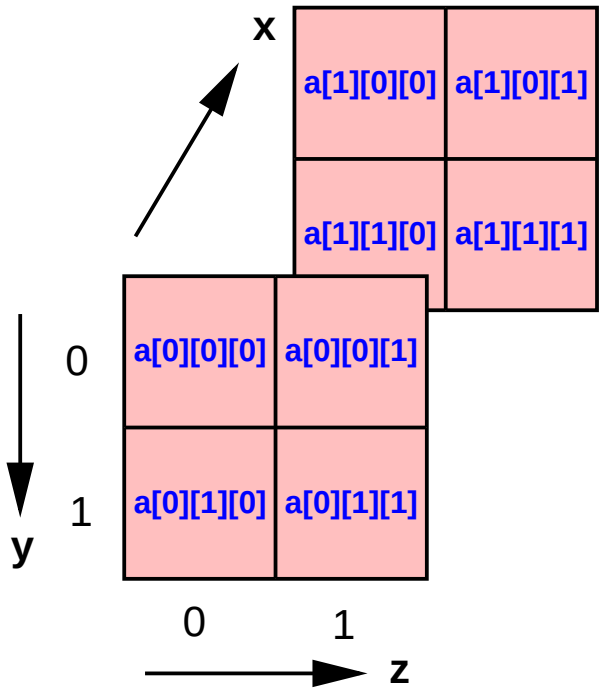
```
#include <iostream>                                                    /* File: 2d-array-fcn.cpp */
#include <cmath>
using namespace std;

float distance(float x1, float y1, float x2, float y2) {
    float x_diff = x1 - x2, y_diff = y1 - y2;
    return sqrt(x_diff*x_diff + y_diff*y_diff);
}

void print_2d_array(const float a[ ][3], int num_rows, int num_columns)
{
    for (int i = 0; i < num_rows; i++) {
        for (int j = 0; j < num_columns; j++) cout << a[i][j] << '\t';
        cout << endl;
    }
}

void compute_all_distances(const float point[ ][2], float dist[ ][3], int num_points)
{
    for (int i = 0; i < num_points; i++)
        for (int j = 0; j < num_points; j++)
            dist[i][j] = distance(point[i][0], point[i][1], point[j][0], point[j][1]);
}

int main(void)
{
    float dist[3][3];                                    // To store distances between any pairs of points
    float point[3][2] = { { 1.0, 1.0 } , { 2.0, 2.0 } , { 4.0, 3.0 } };    // (x, y) coordinates of 3 points
    compute_all_distances(point, dist, 3);
    print_2d_array(dist, 3, 3);
    return 0;
}
```

C++ N-dimensional Array

Syntax: Definition of an N-dimensional Array

<data-type> <array-name> [<size$_1$>] [<size$_2$>] $\cdots$ [<size$_N$>] ;

```
int a[2][2][2] = {1,2,3,4,5,6,7,8}; // sizeof(int) = 4
```

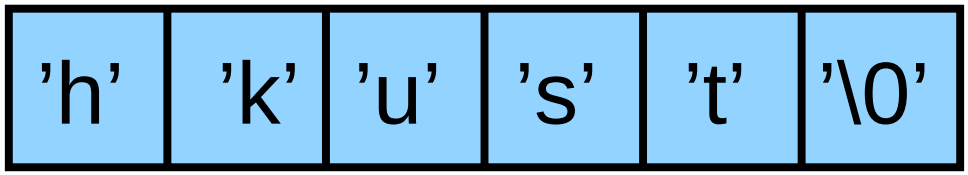


Remarks on Multi-dimensional Array

- Although conceptually a 2D array is like a matrix, and a 3D array is like a cube, the elements of a multi-dimensional array are stored linearly in the memory (just like a 1D array).
- In C++, the elements of a multi-dimensional array are stored in row-major order: row by row.
- There are programming languages (e.g. FORTRAN) that store multi-dimensional array elements in column-major order: column by column.
- In row-major order, the last dimension index runs fastest, while the first dimension index runs slowest.
- If a multi-dimensional array is used in a C++ function, all dimensions other than the first dimension must be specified in its declaration in the function header.

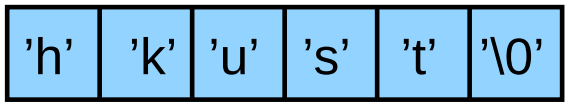
Part III

C String: Special 1D Character Array



C String

- C++ follows C's special way of representing a character string by a 1D character array.
- Just add the null character '\0' (ASCII code = 0) after the last character of the string you need.



- In general, if a string has a length of N, add '\0' at the $(N+1)$th element of the char array.
- The '\0' acts as the end-marker for a character string.
- C++ allows another notation using the double quotes. e.g.

  "hkust" = 'h' 'k' 'u' 's' 't' '\ 0'

Example: C String

```
#include <iostream>                                    /* File: c-string.cpp */
using namespace std;

int main(void)
{
    char s1[6] = {'h', 'k','u','s','t', 'z'};

    /* At this point, s1 is still a simple char array */
    for (int j = 0; j < 5; j++)
        cout << s1[j];
    cout << endl;

    s1[5] = '\0';                                      /* Now, s1 is a C string */
    cout << s1 << endl;

    /* Another notation for initialization, literal constant strings */
    char s2[20] = {'h', 'k','u','s','t', '\0'}; cout << "s2 = " << s2 << endl;
    char s3[20] = "hkust"; cout << "s3 = " << s3 << endl;
    return 0;
}
```

Example: Some C String Functions

```
#include <iostream>                                          /* File: c-string-fcn.cpp */
using namespace std;
const char NULL_CHAR = '\0';

int str_len(const char s[ ])
{
    int j;
    for (j = 0; s[j] != NULL_CHAR; j++)
        ;
    return j;
}

int str_concatenate(const char s1[ ], const char s2[ ], char s[ ])
{
    int j;
    for (j = 0; s1[j] != NULL_CHAR; j++)
        s[j] = s1[j];                                        // Copy s1 to s

    for (int k = 0 ; s2[k] != NULL_CHAR; k++, j++)
        s[j] = s2[k];                                        // Copy s2 after s1
    s[j] = NULL_CHAR;                                        // Make s a C String
    return j;
}

int main(void)
{
    char a[20] = "Albert"; char b[20] = "Einstein"; char c[20]; int length;
    cout << "length of string a = " << str_len(a) << endl;
    cout << "length of string b = " << str_len(b) << endl;
    length = str_concatenate(a, b, c);
    cout << "After concatenation:  " << c << " of length " << length << endl;
    return 0;
}
```

Example: Functions with 2D Character Array

```
#include <iostream>                                    /* File: str-array.cpp */
using namespace std;

void print_strings(const char s[ ][16], int num_of_strings)
{
    for (int j = 0; j < num_of_strings; j++)
        cout << s[j] << "  ";
    cout << endl;
}

int main(void)
{
    // 5 C-strings, each having a max. length of 15 char
    const char word[5][16] = {
        "hong kong",
        "university",
        "of",
        "science",
        "technology"
    };

    print_strings(word, 5);
    return 0;
}
```

Reading C Strings with cin

- cin will skip all white spaces before reading data of the required type until it sees the next white space.
- White spaces are any sequence of ' ', '\t' and '\n'.
- For `char x; cin >> x;`, if the input is " hkust ", cin will skip all the leading white spaces, and gives 'h' to x.
- The same is true for reading a C string.
- For `char x[20]; cin >> x;`, if the input is " hkust ", cin will skip all the leading white spaces, and gives "hkust" to x.
- Thus, cin is not good at reading multiple words or even a paragraph including possibly the newline. Instead, use:

```
cin.getline(char s[], int max-num-char, char terminator);
```

- cin.getline( ) will stop when either (*max-num-char* - 1) characters are read, OR, the terminating character *terminator* is seen. The terminating character is removed from the input stream but is not read into the string.
- The C-string terminating null character is automatically inserted at the end of the read string.

Example: cin.getline( ) from "hacker.txt"

```
#include <iostream>                                    /* File: read-str.cpp */
using namespace std;

int main(void)
{
    const int MAX_LINE_LEN = 1000;
    char s[MAX_LINE_LEN+1];

    // read until the newline character (default)
    cin.getline(s, MAX_LINE_LEN+1, '\n');
    cout << s << endl;

    // read until the character 'W'
    cin.getline(s, sizeof(s), 'W');
    cout << s << endl;

    return 0;
}
```

Example: Palindrome

```
#include <iostream>                                    /* File: palindrome.cpp */
using namespace std;

bool palindrome(char x[ ])
{
    int j = 0;                          // An index reading the array from top (left)
    int k = strlen(x) - 1;              // An index reading the array from bottom (right)

    for ( ; j < k; ++j, --k)
        if (x[j] != x[k])
            return false;
    return true;
}

int main(void)
{
    const int MAX_LINE_LEN = 255;
    char whole_line[MAX_LINE_LEN+1];

    while (cin.getline(whole_line, MAX_LINE_LEN+1, '\n'))
        cout << boolalpha << palindrome(whole_line) << endl;
    return 0;
}
```