Week 5

Muhao Chen
Email: muhaochen@ucla.edu
Outline

- Multi-dimensional arrays
- C-string
Multi-dimensional Array

- An array of arrays
- `int xy[3][4] = { {1,2,3,4} , {5,6,7,8} , {4,3,2,1} } ;`

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

- Two-dimension represents a matrix (2-d tensor)
- Three-dimension represents a cube (3-d tensor)
- Higher-dimension ... hyper-cube (k-d tensor)

- All elements in a multi-dimensional array has to be the same type
How to declare a 2-d array

- **Without initialization**
  - `int xy[3][4]`;
  - means 3 rows of 4-int arrays,
  - Or a 3 rows * 4 cols of int matrix

- **Type name[##rows][##cols]**
  - Both #rows and #cols need to be specified in declaration (**if without initialization**)
  - In the same way of #elements for a 1-d array
How to initialize a 2-d array;

- Regard it as an array of arrays
  - `int xy[3][4] = { {1,2,3,4} , {5,6,7,8}, {4,3,2,1} };`
- Regard it as a series of int folds to a matrix
  - `int xy[3][4] = {1,2,3,4, 5,6,7,8, 4,3,2,1};`
- #rows can be omitted if with initialization
  - `int xy[ ][4] = {1,2,3,4, 5,6,7,8, 4,3,2,1};`
- We’ll get the same 3*4 int array

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Initialize a 2-d array with less elements

- Less elements in some rows
  - int xy[3][4] = {{1,2,3,4}, {5,6}, {4,3,2,1}};
  - Missing elements in such rows will be all-zero

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- Less total elements
  - int xy[3][4] = {1, 2, 3, 6, 7, 8, 4, 3, 2};
  - Elements in the end will be all-zero

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Unacceptable ways of initializing a 2-d array

- int xy[3][4] = {{1,2,3}, {5,6,7,8, 9}, {4,3,2,1}};
  - Row out-of-bound  X

- int xy[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12,13};
  - More elements than #cols * #rows  X

- int xy[3][] = {1,2,3,4,5,6,7,8,9,10,11,12};
  - #cols not specified  X

- int xy[3][4] = {1,2,3,4,”a”,6,7,8,9,10,11,12};  X
  - Inconsistent and inconvertible types of elements
Accessing elements in a 2-d array

Access an element
- `xy[1][2]; // 2nd row, 3rd col`  
  7

Access a row
- `xy[1]; // 2nd row`  
  5  6  7  8

However, there’s no direct way to access a column
Example: Transpose a square matrix

- Swap each element in position (x,y) with that in position (y,x)
- To simplify, make the matrix as a square

```c
void transpose(int m[][], int n) {
    int tmp;
    for (int i=0; i<n; ++i)
        for (int j=i + 1; j < n; ++j) {
            tmp = m[i][j];
            m[i][j] = m[j][i];
            m[j][i] = tmp;
        }
}
```
Note: 2-d Array will not check the bound

- For a 1-d array, we know this will be regarded as out-of-bound
  - int a[3] = {0}; cout << a[3];
- For a 2-d array, the compiler won’t know when it’s out-of-bound

```cpp
int main()
{
    int s[3][4] = {{1}};
    cout << s[4][5];
    //cout << strcmp(t, s) << endl;
    system("pause");
    return 0;
}
```
Note: 2-d Array will not check the bound

- s[4][5] is just to access the \((4 + 1) \times \#\text{cols} + 5\)-th byte after s.

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
</tr>
</tbody>
</table>

The s[3][4] we defined

The s[4][5] we accessed
Note: 2-d Array will not check the bound

- We need to remember the boundaries of 2-d arrays ourselves. Otherwise it’s possible for a 2-d array to access any unexpected block of the memory.
Applications of 2-d Arrays

- Image processing
  - Images are usually represented as k-d arrays

- Chessboard problems
  - 8-queen problem
  - Knight’s tour problem
  - Etc

- Graph
  - (adjacency matrix)

- 2-d tensors for neural networks these days
C-strings in detail
C-String review

- What is a c-string
  - A char array which terminates by ‘\0’ (or 0, or NULL).

- How to initialize a c-string
  - Use either a string value or a set of char ended with a ‘\0’.
    - char c[] = {'g', 'o', 'o', 'g', 'l', 'e', '\0'}
    - char c[] = “google”

- How to input/output a c-string
  - char c[100]; cin >> c; cout << c;

- How to copy a c-string (deep copy)
  - char c[]="google"; char d[100];
  - for (i=0; c[i] != ‘\0’; i++) d[i] = c[i];
  - d[i] = ‘\0’;
C-string

- What if multiple ‘\0’ coexist in a C-string initialization
  - The first ‘\0’ always represents the end
  - char c[100]=“abc\0def\0hg”;
  - cout << c;
  - abc
  - cout << c[4];
  - d
Library functions for C-string

- include `<cstring>` (or include `<string.h>`)
  - Library functions for C-strings
- Member functions of C++ Strings, such as size() and substr(), no longer work for C-strings.
`strlen(s)`

- Returns the length of `s`.
- `char s[] = “aaaaaaaa”;
- `cout << strlen(s);`
- `6`
Implement strlen(s)

- What if we’re not allowed to use strlen

```c
int strlen(s) {
    int len;
    for (len=0; s[len] != '\0'; ++len);
    return len;
}
```
strcpy(t, s)

- Copy the C-string s to C-string t.
- This works as a **deep copy**.
- We have to make sure there’s enough space in t.
  - If length of s is larger than the size of t, program will cause a runtime error.

- The return value is t.
```cpp
#include "stdafx.h"
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char c[100] = "abc";
    char s[2];
    strcpy(s, c);
    cout << s;
    system("pause");
    return 0;
}
```

strcpy(t, s) error: insufficient space in t
**strncpy(t, s, n)**

- Copy at most n characters from s to t.

```c
char* strncpy(char *t, char *s, int n) {
    for (int i=0; i < n; ++i) {
        t[i] = s[i]; if(s[i]) == '\0') break;
    }
    return t;
}
```

- A safe way of strcpy(t, s):
  - `strncpy(t, s, sizeof(t) / sizeof(char));`

- Note: if n < strlen(s), no ‘\0’ will be copied to t!
  - Thus we cannot assume t as a completed C-string by strncpy.
  - We have to manually assign t[n]=‘\0’;
int main ()
{
    char str1[] = "To be or not to be";
    char str2[60];
    char str3[60] = “David the Someberg who buys lots of watermelons”;

    /* copy to sized buffer (overflow safe): */
    strncpy ( str2, str1, sizeof(str2) );
    cout<< str2 << endl;

    /* partial copy (only 5 chars): */
    strncpy ( str3, str2, 5 );
    cout<< str3 << endl;
    str3[5] = '\0'; /* set the null character manually */
    cout<< str3 << endl;
    return 0;
}
strcat(t, s)

- Append C-string s to the end of t.
  - t += s won’t do the job. Use strcat(t, s) instead.

```c
char * strcat(char *t, char *s) {
    int shift = strlen(t)
    for (int i=0; i <= strlen(s); ++i)t[shift + i]=s[i];
    return t;
}
```

- Note: there’s also no size check for t, we have to make sure t has enough space for strlen(t) + strlen(s);
```c
/* strcat example */
#include <stdio.h>
#include <string.h>

int main ()
{
    char str[80] = "";
    strcpy (str,"these ");
    strcat (str,"strings ");
    strcat (str,"strings ");
    strcat (str,"are ");
    strcat (str,"are ");
    strcat (str,"concatenated.");
    cout << str;
    return 0;
}
```

these strings are concatenated.
int strcmp(char *t, char *s)

- Compare two C-strings
  - s == t; s < t; s > t; won’t do the work.
- Return value of strcmp is int, not bool!
  - t equals to s: return 0
  - t less than s: return something <0
  - t greater than s: return something >0
- How to tell if t is greater than s?
  - if (strcmp(t, s) > 0) …
```cpp
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char s[100] = "999999", t[10] = "11123";
    cout << strcmp(t, s) << endl;
    system("pause");
    return 0;
}
```
```c
int main ()
{
    char key[] = "apple";
    char buffer[80];
    do {
        printf ("Guess my favorite fruit? ");
        fflush (stdout);
        cin >> buffer;
    } while (strcmp (key,buffer) != 0);
    cout << "Correct answer!";
    return 0;
}
```

Guess my favourite fruit? orange
Guess my favourite fruit? apple
Correct answer!
## Summary of C-string functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>strlen(s)</td>
<td>Return the length of s</td>
</tr>
<tr>
<td>strcpy(t, s)</td>
<td>Copy s to t.</td>
</tr>
<tr>
<td>strncpy(t, s, n)</td>
<td>Copy at most n characters from s to t.</td>
</tr>
<tr>
<td>strcat(t, s)</td>
<td>Append s to t.</td>
</tr>
<tr>
<td>strcmp(t, s)</td>
<td>Compare s and t.</td>
</tr>
</tbody>
</table>

Convert a C-string to a C++ String

- char cs[10] = “hello”;
- We can use either of the two below:
  - string cpps = cs;
  - string cpps(cs);
Convert a C++-string to a C-string

- string cpps = “abc”; char c[100];
- Don’t use c = cpps; //error
- Use strcpy(c, cpps.c_str()); instead
  - c_str() Get the “C-string body” of a C++ string
Create an Array of C-strings

- A C string is an array of characters. This means an array of C strings is simply a 2D array:
  - char s[10][20];
  - In s, we can store up to 10 C strings, and each C string can be at most 19 characters long.

- s[2] : the third C-string
- s[2][4] : the fifth character of the third C-string
Assign Values to Already-defined C-String Array

- Always use strcpy to store a string to a row.
- char s[10][20];
- strcpy(s[0], “First string”);
- s[0] = “First string”; won’t do the work
Example of creating and using a C-string array

```c
char s[3][6]; // Can store three 5-letter words.
strcpy(s[0], "hello");
strcpy(s[1], "apple");
strcpy(s[2], "world");
cout << s[0] << endl; // prints "hello"
cout << s[2][2] << endl; // prints "r"
```
Thank you.