

C/C++ Programming Language

CS219 Spring

Feng Zheng

Lecture 3



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



Content

- Brief Review
- Compound Types
 - Array
 - String
 - string-class string
 - Structure
 - Pointer
- Managing memory for data
- Summary

Brief Review



Fundamental types

- Integer Type
 - Bits and Bytes
 - Unsigned and signed types
- Char Type
- Floating-point Type
 - Precision
- Arithmetic Operators
 - Conversions



Compound Types



Content

- Arrays
- Array-style strings
- string-class strings
- Structures
- Unions
- Enumerations

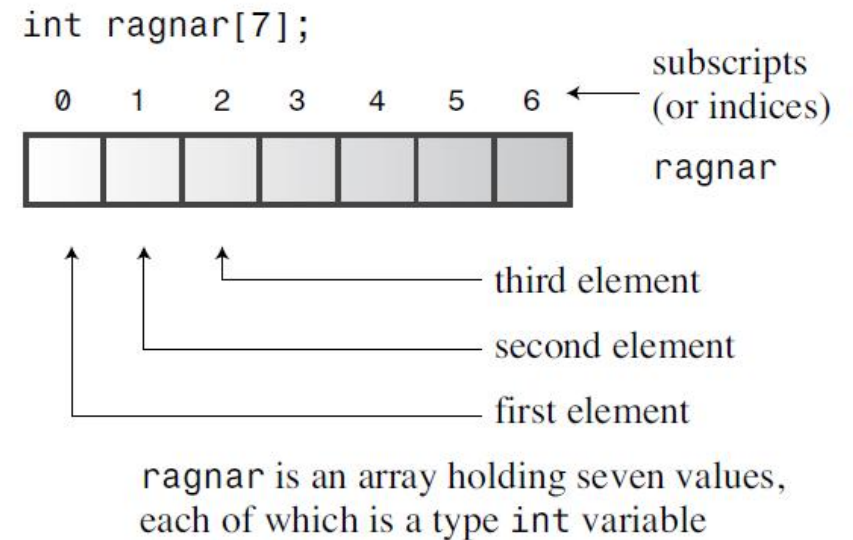
Array



Arrays

- An array is a data form that can hold **several** values, all of **one type**
- To define:
 - The **type** of value to be stored in each element
 - The **name** of the array
 - The **number** of elements in the array must be an **integer** constant, such as **10** or a **const value**, **MICROS**, or a constant **expression**
 - Square brackets []

Why?





Arrays

- Some statements for an array
 - Declaring an array
 - Assigning values to array elements
 - Initializing an array
- Run program
 - `// arrayone.cpp -- small arrays of integers`
 - Note that if you use the `sizeof` operator with an `array name`, you get the number of `bytes` in the `whole array`
 - `First` element index is `0`
 - `Error`: if subscript is equal or greater than the number of elements



Initialization Rules for Arrays

- Several rules about initializing arrays
 - Able to
 - ✓ Use the **initialization** form **only** when defining the array
 - ✓ Use **subscripts** and assign values to the elements of an array individually
 - ✓ **Partially** initialize an array, the compiler sets the **remaining** elements to **zero**
 - Cannot
 - ✓ Use **initialization** later
 - ✓ Assign **one array** wholesale to **another**

```
int cards[4] = {3, 6, 8, 10};    // okay
int hand[4];                    // okay
hand[4] = {5, 6, 7, 9};         // not allowed
hand = cards;                   // not allowed
```

```
float hotelTips[5] = {5.0, 2.5};
long totals[500] = {0};
short things[] = {1, 5, 3, 8};
```



C++11 Array Initialization

- Rules in C++11

- Can **drop** the **=** sign

```
double earnings[4] {1.2e4, 1.6e4, 1.1e4, 1.7e4}; // okay with C++11
```

- **Cannot** convert from a **floating-point** type to an **integer** type(**narrowing**)
- **Cannot** assign **int** type to **char** type (**Outside** the range of a char)

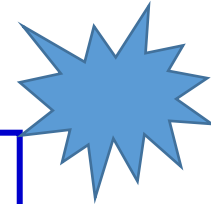
```
long plifs[] = {25, 92, 3.0};           // not allowed
char slifs[4] {'h', 'i', 1122011, '\0'}; // not allowed
char tlifs[4] {'h', 'i', 112, '\0'};     // allowed
```

String



Strings

- A string is a **series** of **characters** stored in **consecutive** bytes of memory
 - **C-style (array)** string
 - string **class** library
- Store a string in an **array** of char (**C-style**)
 - The **last** character of every string is the **null** character
 - This null character is written **\0**
 - The character is with **ASCII** code **0**
 - It serves to **mark** the string's **end**



```
char dog[8] = { 'b', 'e', 'a', 'u', 'x', ' ', 'I', 'I' };           // not a string!  
char cat[8] = { 'f', 'a', 't', 'e', 's', 's', 'a', '\0' };         // a string!
```



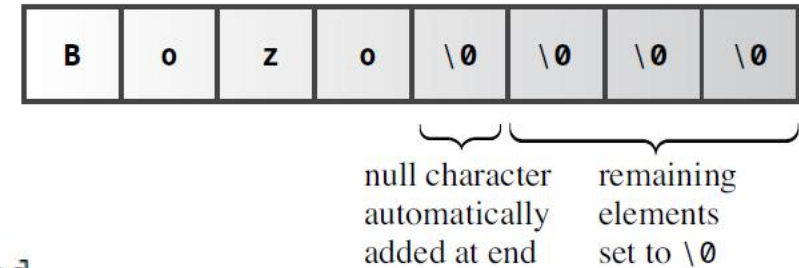
Strings

- Using a **double quoted** string
 - Called a string **constant** or string **literal**
 - Include the **terminating null** character **implicitly**

```
char bird[11] = "Mr. Cheeps";    // the \0 is understood
char fish[] = "Bubbles";        // let the compiler count
```

- Make sure the array is **large** enough to hold **all the** characters
- Note that a string constant (with **double quotes** " ") is **not interchangeable** with a character constant (with single quotes ' ')

```
char boss[8] = "Bozo";
```



```
char shirt_size = 'S';           // this is fine
char shirt_size = "S";          // illegal type mismatch
```

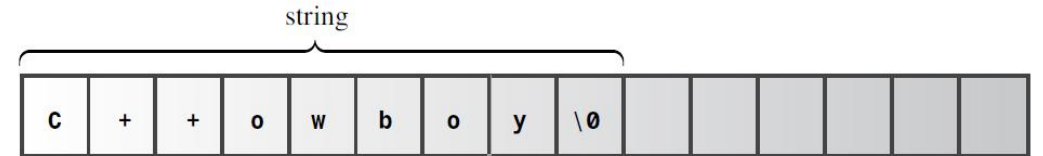


Concatenating String Literals

- C++ enables to **concatenate** string literals
 - Any two string constants separated only by **whitespace**

```
cout << "I'd give my right arm to be" " a great violinist.\n";  
cout << "I'd give my right arm to be a great violinist.\n";  
cout << "I'd give my right ar"  
"m to be a great violinist.\n";
```

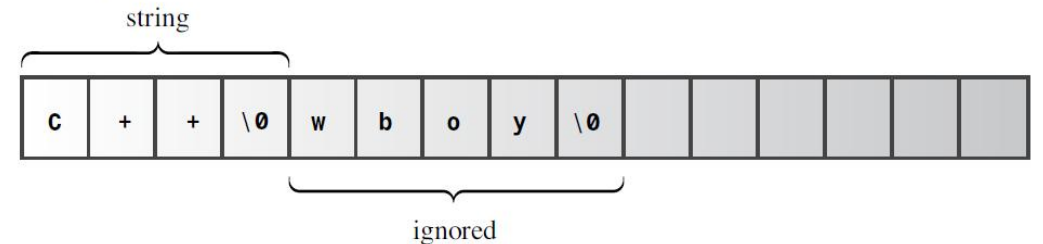
```
const int ArSize = 15;  
char name2[ArSize] = "C++owboy";
```



- **Run program**

- Using Strings in an **Array**
- // strings.cpp -- storing strings in an array

```
name2[3] = '\0';
```



- **Shortening** a string with **\0**
- Beware of memory **overflow (Problem)**

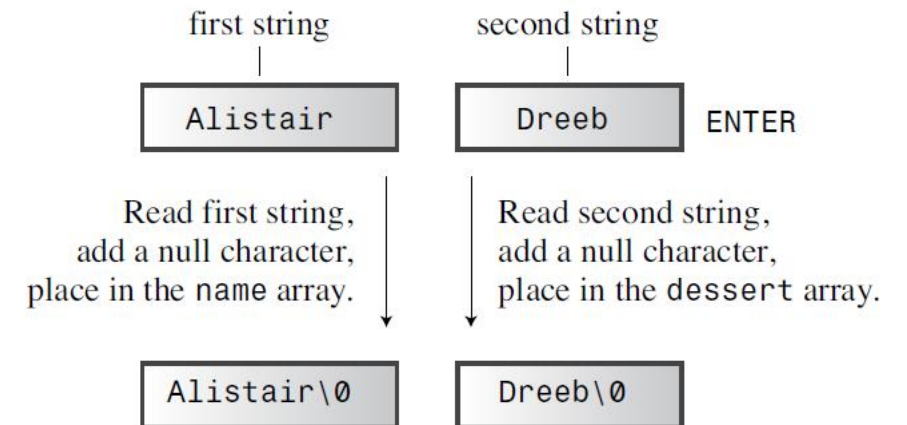


Adventures in String Input

- Run program

- `// instr1.cpp -- reading more than one string`
- The `cin` technique is to use **whitespace**—spaces, tabs, and newlines (`\0`)—to **delineate** a string
- The input string might turn out to be **longer** than the **destination array** (buffer)

- A white space causes a **problem**





Reading String Input a Line at a Time (solved)

- To solve the problem:
- 1. Line-oriented input with **getline()**
- Run program
 - // instr2.cpp -- reading more than one word with getline()
 - **Two** arguments
- 2. Line-oriented input with **get()**
- Run program
 - Read the single next character, even if it is a **newline**
 - // instr3.cpp -- reading more than one word with get() & get()



Mixing String and Numeric Input

- **Problem:** mixing **numeric** input with **line-oriented** string input
- **See a problem in program example**
 - `// numstr.cpp -- following number input with line input`
 - The problem is that when `cin` reads the year, it **leaves** the **newline** generated by the **Enter key** in the input queue. Then `cin.getline()` reads the newline as an **empty** line and assigns a **null** string to the address array
- **Solve it:** **`cin.get()`**;

Code:

```
char name[10];  
cout << "Enter your name: ";  
cin.getline(name, 10);
```

User responds by typing **Jud**, then pressing **ENTER**

Enter your name: Jud **ENTER**

`cin.getline()` responds by reading Jud, reading the newline generated by the Enter key, and replacing it with a null character.

J	u	d	\0						
---	---	---	----	--	--	--	--	--	--

newline replaced with a null character.

string-class strings



string class

- The ISO/ANSI **C++98 Standard** expanded the C++ library
- Include the string **header** file: `#include<string>`
- Run: `// strtype1.cpp -- using the C++ string class`
 - **Initialize** a string object, in a similar way as a C-style string
 - Use **cin** to store keyboard input in a string object
 - Use **cout** to display a string object
 - Use array notation to **access** individual characters stored in a string object
- Differences
 - Treat object as a **simple variable**, not as an array
 - Allow the program to handle the **sizing automatically**



C++11 String Initialization

- C++11 enables 4 kinds of initialization

- Array-style
- String class

- Assign one string object to another

- Array assignment

```
char first_date[] = {"Le Chapon Dodu"};
char second_date[] {"The Elegant Plate"};
string third_date = {"The Bread Bowl"};
string fourth_date {"Hank's Fine Eats"};
```

```
char charr1[20];           // create an empty array
char charr2[20] = "jaguar"; // create an initialized array
string str1;               // create an empty string object
string str2 = "panther";   // create an initialized string
charr1 = charr2;           // INVALID, no array assignment
str1 = str2;               // VALID, object assignment ok
```

- Use the + and += operators

```
string str3;
str3 = str1 + str2;           // assign str3 the joined strings
str1 += str2;                 // add str2 to the end of str1
```



More string Class Operations

- Three functions for **array-style** string
 - strcpy(): **copy** a string to a character array → =
 - strcat(): **append** a string to a character array → +=
 - strlen(): **calculate** the **length** of a character array → `***.size()`
- See three operations in program
 - `// strtype3.cpp -- more string class features`
- Conclusions
 - string objects tends to be **simpler** than using the C string functions
 - string objects tends to be **more safe** than that of the C



More on string Class I/O

- See length of string in program

- `//strtype4.cpp -- line input`

- The difference and problems of array-style string

- `strlen()` reaches a **null character**

- string object is automatically set to **zero size**

- Array-style string has **fixed** size of input

- `cin.getline(charr, 20); // Array-style string`

- `getline(cin, str); // string class`



Other Forms of String Literals

- Beside char, we have **more** following types

- `wchar_t` `wchar_t title[] = L"Chief Astrogator"; // w_char string`
- `char16_t` `char16_t name[] = u"Felonia Ripova"; // char_16 string`
- `char32_t` `char32_t car[] = U"Humber Super Snipe"; // char_32 string`

- **Unicode** characters called UTF-8

- Using `u8` prefix to indicate

- **C++11** adds a raw string

- Delimiter: `"(***)"`
- Using **R prefix** to indicate

Structures, Unions and Enumerations



Introducing Structures

- Why structures?
 - Almost all previous types are those you can **directly use**
 - A structure is a **more versatile** data form than an **array**
 - A structure is a **user-definable type**
- The keyword **struct** → make a **new type**

```
the struct    the tag becomes the name
keyword      for the new type
{
  struct inflatable
  {
    char name[20];
    float volume;
    double price;
  };
};
```

opening and closing braces { }

structure members

terminates the structure declaration



Using a Structure in a Program

- How to create a structure?
 - **Where** to place the structure declaration?
Inside or outside of main
 - Can a structure use a string class member?
Yes
 - Assignment: use a **comma-separated** list of values enclosed in a pair of **braces**
 - In C++11, the = sign is optional
 - **Empty** braces result in the individual members being set to **0**
- See assignment and member access
 - `// structur.cpp -- a simple structure`

external declaration—can be used in all functions in file	_____	<pre>#include <iostream> using namespace std; struct parts { unsigned long part_number; float part_cost; }; void mail(); int main() { struct perks { int key_number; char car[12]; }; parts chicken; perks mr_blug; ... } void mail() { parts studebaker; ... }</pre>
local declaration—can be used only in this function	_____	
type parts variable	_____	
type perks variable	_____	
type parts variable	_____	
can't declare a type perks variable here	_____	



Other Structure Properties

- What **actions** you can do for structures?

- Pass structures as **arguments (multiple)** to a function
- Have a function use a structure as a **return value (multiple)**
- Combine the definition of a structure form with the **creation of structure variables**
- Have member **functions** in addition to member variables



- Run program

- `// assgn_st.cpp -- assigning structures`
- **Member-wise assignment**: use the assignment operator (=) to assign one structure to another of the same type

```
struct Books {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
} book;
```



More Structure Properties: Array

- **Arrays** of Structures

- Create arrays whose **elements are structures**
- An example
 - ✓ gifts itself is an array, **not** a **structure**
 - ✓ gifts[0] is a **structure**

```
inflatable gifts[100]; // array of 100 inflatable structures

cin >> gifts[0].volume; // use volume member of first struct
cout << gifts[99].price << endl; // display price member of last struct

inflatable guests[2] = // initializing an array of structs
{
    {"Bambi", 0.5, 21.99}, // first structure in array
    {"Godzilla", 2000, 565.99} // next structure in array
};
```



Unions

- A union is a data format
 - Can hold **different** data types but **only one** type **at a time**
 - Can use two or more formats but **never** simultaneously
 - **Save** memory
 - **union** Keyword → make a **new type**



```
union one4all
{
    int int_val;
    long long_val;
    double double_val;
};

one4all pail;
pail.int_val = 15;           // store an int
cout << pail.int_val;
pail.double_val = 1.38;     // store a double, int value is lost
cout << pail.double_val;
```



Enumerations

- The C++ enum facility provides an alternative to **const** for creating symbolic constants (**#define**)

- enum **spectrum** {red, orange, yellow, green, blue, violet};
 - ✓ It makes spectrum the name of a **new type**
 - ✓ It establishes the members as symbolic constants for the integer values **0-5**
- By default, enumerators are assigned integer values **starting with 0** for the first enumerator, 1 for the second enumerator, and **so forth**
- The assigned values **must be integers**
- **enum** Keyword → make a **new type**





Enumerations

```
enum spectrum {red, orange, yellow, green, blue, violet, indigo, ultraviolet};
```

- What **operations** can you do for enumerations?

- **Assign it** using the member
- You can set enumerator values **explicitly**

```
enum bits{one = 1, two = 2, four = 4, eight = 8};
```

- **Assign other** variables using it
- **Typecast** values within the range
- Beware of the value **ranges** for enumerations

```
spectrum band; // band a variable of type spectrum

band = blue;    // valid, blue is an enumerator
band = 2000;    // invalid, 2000 not an enumerator

band = orange;  // valid
++band;         // not valid, ++ discussed in Chapter 5
band = orange + red; // not valid, but a little tricky

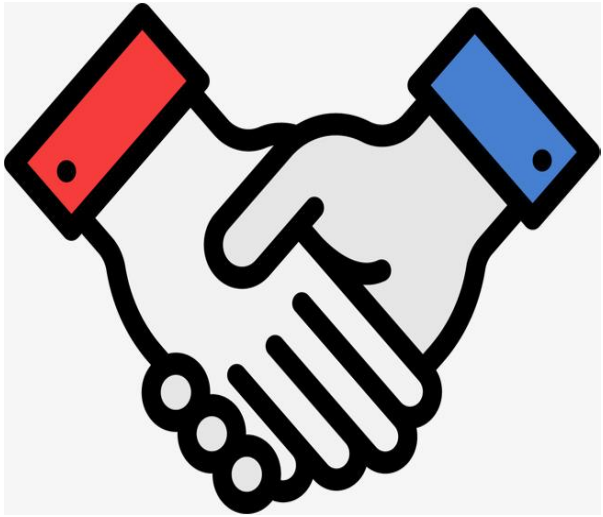
int color = blue; // valid, spectrum type promoted to int
band = 3;         // invalid, int not converted to spectrum
color = 3 + red;  // valid, red converted to int
band = spectrum(3); // typecast 3 to type spectrum
band = spectrum(40003); // undefined
```

```
band = orange + red; // not valid, but a little tricky
```




Summary

- Compound types
 - Array
 - Array-style string
 - String class
 - Structure



Thanks



zhengf@sustech.edu.cn