第五讲

Strings, Streams

薛浩

2023 年 3 月 28 日

www.stickmind.com

今日话题

话题 1：编程基础 初学编程的新手，一般应该熟练使用函数和库处理字符串相关的编程任务。

话题 2：抽象数据类型的使用 在尝试实现抽象数据类型之前，应该先熟练使用这些工具解决问题。

话题 3：递归和算法分析 递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

话题 4：类和内存管理 使用 C++ 实现数据抽象之前，应先学习 C++ 的内存机制。

话题 5：常见数据结构和算法 在熟练使用抽象数据类型解决常见问题之后，学习如何实现它们是一件很自然的事情。

话题 1：编程基础

初学编程的新手，一般应该熟练使用函数和库处理字符串相关的编程任务。

- C++ 基础
- 函数和库
- 字符串和流

Figure 1: 语音算法

计算机如何处理文本信息？

目录

复习：函数

Swap 交换函数

以下函数能否交换两个值呢?

```
void swap(int x, int y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int value1 = 3, value2 = 4;
swap(value1, value2);
cout << "The value #1 is " << value1 << endl;
cout << "The value #2 is " << value2 << endl;
```

Call by Reference 引用传递

C++ 支持引用传递参数，允许函数和主调函数共享数据。引用参数需要在参数后面追加 & 符号：

```
void swap(int & x, int & y);
void swap(int & x, int & y) {
  int tmp = x;
  x = y;
  y = tmp;
}
```

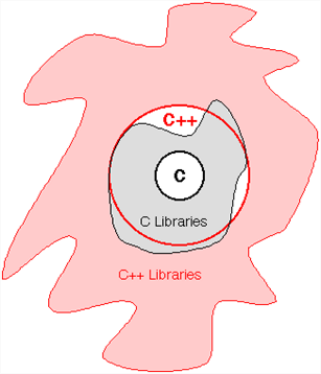
引用参数的两个主要作用是：

- 函数需要返回多个值时，可以通过参数列表在主调函数/被调函数之间共享信息
- 对于大型数据对象，例如 GObject，可以避免拷贝带来的性能消耗

C++ 和 C 的关系

C++ 和 C 的关系

C++ 成功背后的部分原因是包含了 C 的子集。这样的设计策略使得 C 程序可以增量式地转向 C++。
这个策略的缺点也非常明显，造成了 C++ 设计上的不连贯性。



Characters 字符

ASCII

b_7 b_6 b_5 → / Bits b_4 b_3 b_2 b_1 ↓	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Column → / Row ↓	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	—	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	__	o	DEL

cctype

`bool isdigit(char ch)` Determines if the specified character is a digit.
`bool isalpha(char ch)` Determines if the specified character is a letter.
`bool isalnum(char ch)` Determines if the specified character is a letter or a digit.
`bool islower(char ch)` Determines if the specified character is a lowercase letter.
`bool isupper(char ch)` Determines if the specified character is an uppercase letter.
`bool isspace(char ch)` Determines if the specified character is ***whitespace*** (spaces and tabs).
`char tolower(char ch)` Converts `ch` to its lowercase equivalent, if any. If not, `ch` is returned unchanged.
`char toupper(char ch)` Converts `ch` to its uppercase equivalent, if any. If not, `ch` is returned unchanged.

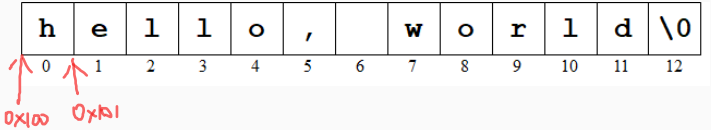
cctype

Strings 字符串

C 字符串

现代语言大都会使用到字符串数据，哪怕只是向用户显示一些指令或打印结果。

从概念上讲，字符串只是一个字符数组，这也是 C++ 的 C 子集中实现字符串的方式。在一串字符周围加上双引号，就能得到 C 字符串，其中字符存储在一个字节数组中，以 ASCII 值为 0 的空字节结尾。例如，C 字符串 “hello, world”中的字符排列如下：



字符串中的字符位置用索引编号标记，该索引从 0 开始并延伸到比字符串长度小 1 的位置。

因为 C++ 包含了 C 字符串所有内容，所以必须学会识别这种字符串的存在。

C++ 字符串

在写程序时，使用 C++ 的 string 类会容易得多，该类将字符串实现为抽象数据类型，由其行为而不是表示来定义。

所有使用 string 类的程序都必须包含 string 库接口，并且类型名要添加标准库命名空间：

```
#include <string>
std::string s;
```

由于 string 是一个类，我们同样可以使用消息传递的语法对 string 变量进行操作。

```
int len = s.length();
```

string 类方法

`str.length()` Returns the number of characters in the string **`str`**.
`str.at(index)` Returns the character at position **`index`**; most clients use **`str[index]`** instead.
`str.substr(pos, len)` Returns the substring of **`str`** starting at **`pos`** and continuing for **`len`** characters.
`str.find(ch, pos)` Returns the first index ≥ **`pos`** containing **`ch`**, or **`string::npos`** if not found.
`str.find(text, pos)` Similar to the previous method, but with a string instead of a character.
`str.insert(pos, text)` ←*Destructively changes* **`str`** Inserts the characters from **`text`** before index position **`pos`**.
`str.replace(pos, count, text)` ←*Destructively changes* **`str`** Replaces **`count`** characters from **`text`** starting at position **`pos`**.

string

运算符重定义

和很多现代语言一样，C++ 也允许重新定义运算符的含义。因此，许多字符串的操作都是以运算符的形式存在的，比如 + 用于连接两个字符串。

`str[i]` Returns the i^{th} character of **`str`**. Assigning to **`str[i]`** changes that character.
`s1 + s2` Returns a new string consisting of **`s1`** concatenated with **`s2`**.
`s1 = s2;` Copies the character string **`s2`** into **`s1`**.
`s1 += s2;` Appends **`s2`** to the end of **`s1`**.
`s1 == s2` (and similarly for **`<`**, **`<=`**, **`>`**, **`>=`**, and **`!=`**) Compares to strings lexicographically.

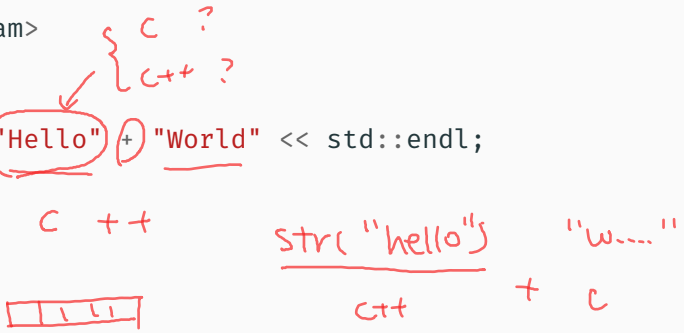
练习：连接字符串

以下代码的输出是什么？

```
include <iostream>

int main() {
  std::cout << "Hello" + "World" << std::endl;

  return 0;
}
```



练习: 修改字符串

以下代码的输出是什么?

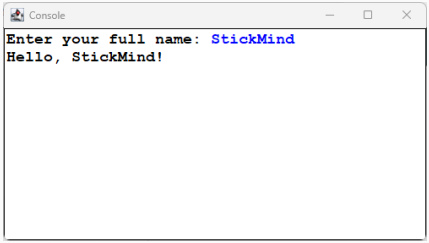
```
void gollyGee(string&text) {
    text[0] = 'g';
}

int main() {
    string message = "wiggle";
    gollyGee(message);
    cout << message << endl; // <-- Here
    return 0;
}
```

练习: HelloName

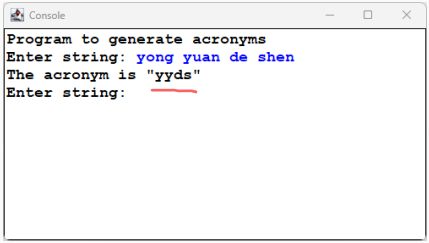
修改 HelloWorld 程序，获取用户输入并打印不同问候语：

Hint: 学习使用 simpio.h 中的 getLine 接口



练习: Acronym

首字母缩略词（Acronym）是通过按顺序提取每个单词的第一个字母而形成的单词，例如“self-contained underwater breathing apparatus” → “scuba”



Data Files 数据文件

Data Files 数据文件

文件（file）通常用于存储永久数据对象。在大多数情况下，文件存储在硬盘上，但也可以存储在可移动介质上，如 CD 或闪存驱动器。

文件可以包含许多不同类型的信息。例如，当编译 C++ 程序时，编译器会将源码文件转换成二进制表示的对象文件。

最常见的文件类型是**文本文件**（text file），它包含字符串中的字符数据。

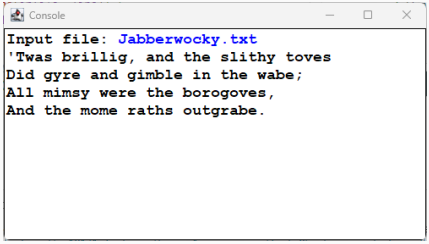
文本文件 vs 字符串

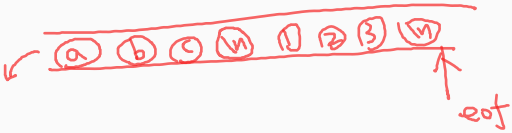
文本文件和字符串都包含字符数据，但两者之间有些重要区别：

存储在文件中的信息是永久性的 字符串变量的值只在变量存在的时间内有效。当方法返回时，局部变量会消失，当对象离开代码块时，实例变量会消失。存储在文件中的信息一直存在，直到文件被删除为止。

文件通常是按顺序读取的 从文件中读取数据时，通常从头开始，按顺序读取字符，可以逐个字符读取，也可以按行读取。一旦读取了一组字符，就继续下一组字符直到文件的末尾。

练习: ShowFileContents





使用文本文件

使用 C++ 读取文本文件，我们需要遵循以下几个步骤：

打开文本文件 构造一个 ifstream 文件流对象，通过调用 open 方法打开文本文件。由于历史原因，open 方法的参数是 C 字符串而不是 C++ 字符串。

读取文本数据 调用 ifstream 类提供的方法，按顺序从文件中读取字符数据。本节课还会介绍如何使用 getline 函数逐行读取文件。

关闭文本文件 通过调用流的 close 方法关闭文件。

TODO: ofstream

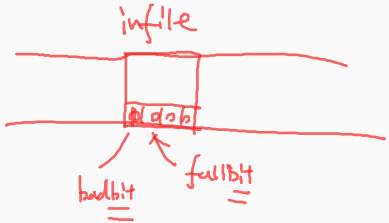
cin → getInteger

buffer

打开文本文件

打开文本文件 构造一个 ifstream 文件流对象，通过调用 open 方法打开文本文件。由于历史原因，open 方法的参数是 C 字符串而不是 C++ 字符串。

```
#include <fstream>
ifstream infile;
infile.open("Jabberwocky.txt");
// or
string filename = "Jabberwocky.txt";
infile.open(filename.c_str());
```



流的状态

使用文件流对象打开文本文件时，并不总会成功。一个健壮的程序需要检测流对象是否有效。以下方法适用于所有流：

stream.fail()	Check whether either failbit or badbit is set
stream.eof()	Check whether eofbit is set
stream.good()	Check whether state of stream is good
stream.clear()	Set error state flags

通常可以直接将流对象用作条件结构的状态表达式：

```
if (stream) ...
// same as
if (!stream.fail()) ...
```

Function Detail

```
bool openFile(ifstream & stream, string filename);
bool openFile(ofstream & stream, string filename);
```

Opens the filestream **stream** using the specified **filename**. This function is similar to the **open** method of the stream classes, but uses a C++ **string** object instead of the older C-style string. If the operation succeeds, **openFile** returns **true**; if it fails, **openFile** sets the failure flag in the stream and returns **false**.

Usage:

```
if (openFile(stream, filename)) ...
```

```
string promptUserForFile(ifstream & stream, string prompt = "");
string promptUserForFile(ofstream & stream, string prompt = "");
```

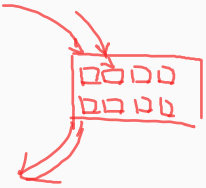
Asks the user for the name of a file. The file is opened using the reference parameter **stream**, and the function returns the name of the file. If the requested file cannot be opened, the user is given additional chances to enter a valid file. The optional **prompt** argument provides an input prompt for the user.

读取文本数据

读取文本数据 调用 ifstream 类提供的方法，按顺序从文件中读取字符数据。

文件流支持单字符读取，以下代码是读取一个文件所有字符的一般模式：

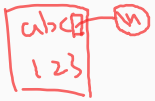
```
char ch;
while (infile.get(ch)) {
  // Perform some operation on the character.
}
// or
while ((ch = infile.get()) != EOF) {
  // Perform some operation on the character.
}
```



读取文本数据

利用函数 getline 可以从流中按行读取数据。以下写法将文件流 infile 中的一行字符写入 line 字符串中（注意，行尾的换行字符会被丢弃）：

```
std::string line;
std::getline(infile, line)
```



函数 getline 的返回值依然是 infile，所以可以直接将函数表达式当作条件状态处理：

```
int max = 0;
std::string line;
while (std::getline(infile, line)) {
    if (line.length() > max) max = line.length();
}
```

abc
123

cout << line << endl;

关闭文本文件

关闭文本文件 通过调用流的 close 方法关闭文件。

```
infile.close();
```

计算机如何处理文本信息？

总结


Stream
strlib.h
File
Call by reference
cctype
String
Character
filelib.h

问题?