

习题一

编写一个函数：

```
1 string replaceA11(string str, char c1, char c2);
```

它返回一个str的复制字符串，其中str中每一个c1都用c2代替。例如，调用replaceA11("nannies", 'n', 'd');

应该返回“daddies”。

```
1 #include <cassert>
2 #include <cstddef>
3 #include <string>
4 #include <iostream>
5 using namespace std;
6
7 string replaceAll(const string &str, char c1, char c2) {
8     string result = str;
9     size_t found = 0;
10
11     while (true) {
12         found = result.find(c1, found);
13         if (found == string::npos) break;
14         result.at(found) = c2;
15     }
16     return result;
17 }
18
19 int main(){
20     assert(replaceAll("nannies", 'n', 'd') == "daddies");
21     cout << "test success!" << endl;
22     return 0;
23 }
```

一旦你编写和测试完这个函数，编写一个重载的函数版本：

```
1 string replaceA11(string str, string s1, string s2);
```

该函数将str中每一个s1字符串都用s2字符串代替。

```
1 #include <cassert>
2 #include <cstddef>
3 #include <string>
4 #include <iostream>
5 using namespace std;
6
7 string replaceAll(const string &str, char c1, char c2) {
8     string result = str;
```

```

9     size_t found = 0;
10
11     while (true) {
12         found = result.find(c1, found);
13         if (found == string::npos) break;
14         result.at(found) = c2;
15     }
16     // 针对字符的特殊处理方式，仅适用于字符，而上述的方式可通用字符串与字符。
17     // 由于c++的问题，处理字符的情景时需要 replaceAll(str, (string)c1, (string)c2)
18     // for (auto &ch: result) {
19     //     if (ch == c1) {
20     //         ch = c2;
21     //     }
22     // }
23     return result;
24 }
25
26 //实现思路:
27 // 使用str.find() 查找可能的 pos，之后利用replace进行替换。
28 string replaceAll(const string &str, const string &s1, const string &s2) {
29     string result = str;
30     size_t found = 0;
31
32     while (true) {
33         found = result.find(s1, found);
34         if (found == string::npos) break;
35         result.replace(found, s1.length(), s2); // 注意是将s1替换成s2，因此使用的是
s1的长度
36     }
37     return result;
38 }
39
40 int main(){
41     assert(replaceAll("nannies", "na", "d") == "dnnies");
42     cout << "test success!" << endl;
43     return 0;
44 }

```

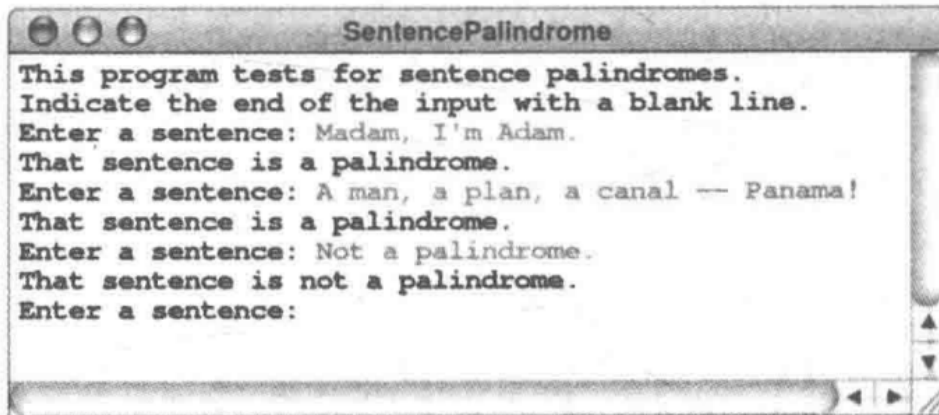
使用了replace与find的API，记住find的形式。

习题二

通过忽略标点符号和字母大小写的差异，回文的概念通常被扩展到整个句子。例如以下句子：Madam, I'm Adam.

这是一个回文的句子，因为如果你只看字母并忽略大小写字母的区别，顺序读和倒序读是一样的。

编写一个判断函数 `isSentencePalindrome(str)`，如果字符串 `str` 和回文句子的定义相匹配，则该函数返回 `true`。你应该能够用该函数去编写一个主程序并产生以下的输出结果：



```
1  #include <cctype>
2  #include <string>
3  #include <iostream>
4  using namespace std;
5
6  bool isPalindrome(const string &rhs) {
7      // 简洁实现:
8      // return rhs == rhs.reserve();
9      size_t len = rhs.length();
10     if (len < 2) {
11         return true;
12     } else {
13         char start = rhs[0];
14         char finish = rhs[len - 1];
15         return start == finish &&
16             isPalindrome(rhs.substr(1, len - 2));
17     }
18 }
19
20 string normalizeSentence(const string &rhs) {
21     // 牢记这种对单个字符处理，并且生成新字符串的方式。
22     string result;
23     for (auto ch : rhs) {
24         if (isalpha(ch)) {
25             result += tolower(ch);
26         }
27     }
```

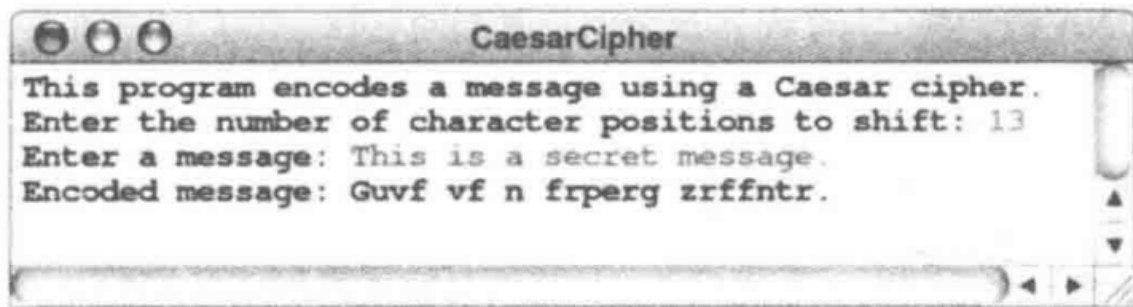
```
28     return result;
29 }
30
31 // 实现思路:
32 // 将普通的句子进行normalize: "Madam, I'm Adam" ==> "madamimadam"
33 // 之后将规格化的句子进行回文判断。
34 bool isSentencePalindrome(const string &rhs) {
35     return isPalindrome(normalizeSentence(rhs));
36 }
37
38 int main() {
39     cout << "This program tests for sentence palindrome." << endl
40         << "Indicate the end of the input with a blank line."<< endl;
41
42     while (true) {
43         cout << "Enter a sentence : " << endl;
44         string line;
45         getline(cin, line); // 使用getline读取一整行
46         if (line.empty()) break;
47
48         if (isSentencePalindrome(line)) {
49             cout << "That sentence is a palindrome." << endl;
50         } else {
51             cout << "That sentence is not a palindrome." << endl;
52         }
53     }
54     return 0;
55 }
```

习题三-五

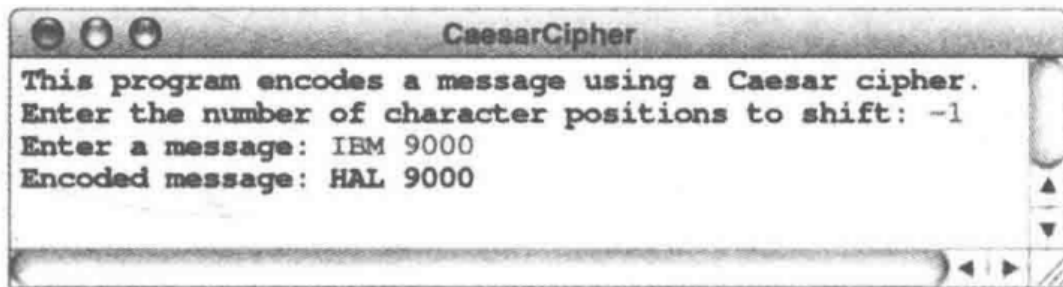
习题三

如果你在童年玩过密码，很有可能在某些地方用过循环码 (cyclic cipher)，它又经常被称为凯撒密码 (Caesar cipher)，因为罗马历史学家苏维托尼乌斯记录到盖乌斯·尤利乌斯·凯撒使用过这项技术，运用这项技术你可以将原始消息中的每个字母用字母表中出现在它前面一个固定距离的字母来代替。

作为一个例子，假设你想通过将每个字母向前移动三个位置来编码一个消息。运用这个密码技术，每个A变为D，B变为E，以此类推。如果你到达了字母表的末尾，处理周期又回到开头，所以X变成A，Y变成B，Z变成C。为了实现一个凯撒密码，**你首先应该定义以下函数：** `string encodeCaesarCipher(string str,int shift);` **该函数返回一个新的字符串，它是将str中的每个字母转换成与它相隔shift处的字母形成的，如果有必要，循环回到字母表的开头。**在你实现了encodeCaesarCipher函数后，编写程序产生以下示例的运行结果：



注意到翻译只适用于字母，其他任何字符都不经修改输出。此外，字母大小写不产生影响：即大小写字母按原样输出。你也应该这样编写你的程序，一个负的shift值意味着字母向着字母表开头的方向进行移动，而不是向结尾方向，正如以下示例的运行结果所表明的：



```
1  #include <cctype>
2  #include <string>
3  #include <iostream>
4  #include <cassert>
5  using namespace std;
6
7  // 核心代码在于加密部分
8  // ch = 'a' + (ch - 'a' + shift + 26) % 26;
9  // 实际逻辑是 ch = start_pos + true_offset;
10 // 1. 使用 %26 保证结果总在A~Z之间取值
```

```

11 // 2. 计算 true_offset 采用 +26，是通过补码的原则来获取，将减法转变成对应的加法。同时不影响shift为正的情景。使其满足，一个负的shift值意味着字母向着字母表开头的方向进行移动，而不是向结尾方向。
12 string caesarCipher(const string &rhs, int shift) {
13     string result = rhs;
14     for (auto &ch : result) {
15         if (isalpha(ch)) {
16             if (isupper(ch)) {
17                 ch = 'A' + (ch - 'A' + shift + 26) % 26; // encoding
18             } else {
19                 assert(islower(ch));
20                 ch = 'a' + (ch - 'a' + shift + 26) % 26; // encoding
21             }
22         }
23     }
24     return result;
25 }
26
27 int main() {
28     cout << "This program encodes a message using a Caesar cipher." << endl
29         << "Enter the number of character positions of shift: ";
30     int shift;
31     cin >> shift;
32     cin.ignore();
33     string line;
34     cout << "Enter a message: ";
35     getline(cin, line);
36     cout << "Encoded message: " << caesarCipher(line, shift);
37     return 0;
38 }

```

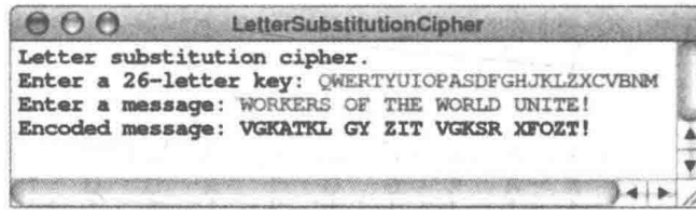
习题四

尽管它们确实很简单，凯撒密码也极易被破解。毕竟，只有25个字符可移动。如果你想破解一个凯撒密码，你只需尝试所有25种可能，然后观察哪一种翻译能将原始的消息转换成可读的。一个更好的策略是原始消息中的每个字母都被任意字母代替而不是被原始消息中相隔固定距离的字符代替。这样的话，编码运算的关键是一个翻译表，它展示了26个字母中每一个改变后的加密形式。这样的一个编码策略称作字母替换密码 (letter-substitution cipher)。

字母替换密码的关键是一个由26个字符构成的字符串，该字符串依次指出了字母表中每个字符的翻译。例如，关键字“QWERTYUIOPASDFGHJKLZXCVBNM”表示编码过程应该使用下面的翻译规则：

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |

采用字母替换密码编写一个能实现加密的程序。该程序应能够复制以下示例中的运行结果：



非常简单，核心与习题2的normalization程序类似，都是对单个字符进行逐一处理。

```

1  #include <cassert>
2  #include <cctype>
3  #include <string>
4  #include <iostream>
5  using namespace std;
6
7  // 计算offset, 并且在cipher_table中找到并替换。
8  string letterSubstitutionCipher( const string& message,
9                                   const string &cipher_table) {
10     string result;
11     for (auto ch : message) {
12         if (isalpha(ch)) {
13             int offset = ch - 'A';
14             assert(offset > 0); // 防御!
15             result += cipher_table[offset];
16         } else {
17             result += ch;
18         }
19     }
20     return result;
21 }
22
23 int main() {
24
25     cout << "Letter substitution cipher." << endl;
26     string ct, msg;
27     cout << "Enter a 26-letter key: ";
28     getline(cin, ct);
29     assert(ct.length() == 26); // 健壮性?
30
31     cout << "Enter a message: ";
32     getline(cin, msg);
33     cout << "Encoded message: " << letterSubstitutionCipher(msg, ct);
34     return 0;
35 }

```

习题五



You

我的意思是，我不喜欢的地方在于使用了find，我觉得str.find低效，有没有可以替代find方法查找获取offset的方法。

Which response do you prefer?

Your choice will help make ChatGPT better.



Response 1

如果你想避免使用 `find()` 函数，可以考虑使用一个数组或映射来记录密钥中每个字符对应的位置。这样，在加密过程中，你可以根据字符的 ASCII 值或者字符本身作为索引来获取对应的偏移量。

下面是一个示例代码，使用一个数组来记录密钥中每个字符对应的位置：



Response 2

如果你希望避免使用 `find()` 方法，一种更高效的方法是使用一个映射表，将密钥中的字符映射到字母表中的位置。你可以使用一个 `std::unordered_map` 或者 `std::map` 来实现这个映射表，将密钥中的字符作为键，其在字母表中的位置作为值。

下面是一个使用 `std::unordered_map` 的示例：

人生第一次触发这个。 两个回答都正确，即避免线性搜索，将 $O(n^2)$ 变成 $O(1)$ 。

```
1  #include <cassert>
2  #include <cctype>
3  #include <cstdint>
4  #include <string>
5  #include <iostream>
6  using namespace std;
7
8  string invertKey(const string& secret_msg, const string &key) {
9      string plaintext_msg;
10     for (auto ch : secret_msg) {
11         if (isalpha(ch)) {
12             size_t offset = key.find(ch); // 避免线性搜索的方案是使用一个映射表
13             // 上述查找与外层循环合力，导致该程序变成了 $O(n^2)$ 。
14             assert(offset != string::npos);
15             plaintext_msg += 'A' + offset;
16         } else {
17             plaintext_msg += ch;
18         }
19     }
20     return plaintext_msg;
21 }
22
23 int main() {
24
25     cout << "Letter substitution cipher." << endl;
26     string key = "QWERTYUIOPASDFGHJKLZXCVBNM";
27     string msg = "VGKATKL GY ZIT VGKSR XFOZT!";
28     // cout << "Enter a 26-letter key: ";
29     // getline(cin, ct);
30     // assert(ct.length() == 26);
31
32     // cout << "Enter a message: ";
33     // getline(cin, msg);
34     cout << "Encoded message: " << invertKey(msg, key);
```



```
35     return 0;
36 }
```

优化版本：通过array构建map结构

```
1  #include <iostream>
2  #include <string>
3  #include <cassert>
4  #include <array>
5  #include <algorithm> // for std::fill
6
7  std::string invertKey(const std::string& secret_msg, const std::string &key) {
8      std::string plaintext_msg;
9      array<int, 26> offset; // 用于记录密钥中每个字符的位置
10
11     std::fill(offset.begin(), offset.end(), -1); // -1 表示当前字符不在密钥中
12
13     // 记录密钥中每个字符的位置
14     for (int i = 0; i < key.size(); ++i) {
15         // char ch = std::toupper(key[i]); // 统一转换为大写字母
16         offset[ch - 'A'] = i; // 字符对应的位置为索引值
17     }
18
19     // 根据密钥中的位置获取明文字符
20     for (char ch : secret_msg) {
21         if (std::isalpha(ch)) {
22             char upperCh = std::toupper(ch);
23             int pos = offset[upperCh - 'A'];
24             assert(pos != -1); // 检查是否找到了对应的位置
25             plaintext_msg += 'A' + pos;
26         } else {
27             plaintext_msg += ch;
28         }
29     }
30
31     return plaintext_msg;
32 }
```

使用自带的unordered_map

```
1  #include <iostream>
2  #include <string>
3  #include <unordered_map>
4  #include <cassert>
5
6  std::string invertKey(const std::string& secret_msg, const std::string &key) {
7      std::string plaintext_msg;
8      std::unordered_map<char, size_t> charMap; // 字符到位置的映射
9  }
```

```
10 // 构建字符到位置的映射
11 for (size_t i = 0; i < key.size(); ++i) {
12     charMap[key[i]] = i;
13 }
14
15 // 根据映射获取明文字符
16 for (char ch : secret_msg) {
17     if (std::isalpha(ch)) {
18         char upperCh = std::toupper(ch);
19         auto it = charMap.find(upperCh);
20         assert(it != charMap.end()); // 检查是否找到了对应的位置
21         plaintext_msg += 'A' + it->second;
22     } else {
23         plaintext_msg += ch;
24     }
25 }
26
27 return plaintext_msg;
28 }
```