Module04-06
C++ 标准库：字符串

C++ 编程语言 - 标准库

- 数据结构简介
- 标准容器
- 常用算法简介
- 标准算法与函数对象
- 迭代器
- → 字符串
- I/O 流
- 数值

标准库 - 字符串

- 字符串 (String)
  - 关于字符串类
  - 常用操作
  - C 标准库字符串函数

字符串 - 关于字符串类

- 字符特征 (char_traits)
  - 字符特征类是将字符类型的差异与字符串操作分离的一种方式
  - 涉及到字符的比较、字符的 int 型值、从 int 值转化为字符等等操作都由 class char_traits 完成

字符串 - 关于字符串类

- 构造函数

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    inline basic_string();
    explicit basic_string(const Alloc& a);
    basic_string(const basic_string& str);
    basic_string(const basic_string& str, size_type pos,
size_type n = npos);
    basic_string(const basic_string& str, size_type pos,
size_type n, const Alloc& a);
    basic_string(const CharT* s, size_type n, const Alloc& a =
Alloc());
    basic_string(const CharT* s, const Alloc& a = Alloc());
    basic_string(size_type n, CharT c, const Alloc& a =
Alloc());
    template<class InputIterator>
    basic_string(InputIterator beg, InputIterator end, const
Alloc& a = Alloc());
    ~basic_string();
};
```

字符串 - 关于字符串类

- 赋值操作

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    // ...
    basic_string& operator=(const basic_string& str);
    basic_string& operator=(const CharT* s);
    basic_string& operator=(CharT c);

    basic_string& assign(const basic_string& str);
    basic_string& assign(const basic_string& str, size_type
pos, size_type n);
    basic_string& assign(const Traits* s, size_type n);
    basic_string& assign(const Traits* s);
    basic_string& assign(size_type n, Traits c);
    template<class InputIterator>
    basic_string& assign(InputIterator first, InputIterator
last);
    //...
};
```

字符串 - 关于字符串类

- 元素的访问
  - 通过下标访问（不检查数组越界）
  - 通过成员函数 at() 访问（检查数组越界）
  - 通过迭代器访问

```
void f() {
    string s("hello world!");
    char c1 = s[1]; // const char& operator[](size_t n) const
    s[2] = 'T'; // char& operator[](size_t n)
    char c2 = s.at(3); // const char& at(size_t n) const
    s.at(5) = '^'; // char& at(size_t n)
    string::const_iterator it = s.begin();
    for (; it != s.end(); ++it)
        cout << *it;
    cout << endl;
}
```

字符串 - 关于字符串类

- npos
  - basic_string 中定义的静态数据成员，通常是为该类型 (size_t) 最大的值，用来表示字符串操作的返回结果，如果是 nops 表示此次操作未果

```
static const size_type npos = static_cast<size_type>(-1);
```

- 错误和异常
  - 通过成员函数 at() 访问字符串元素，如果下标越界，抛出 out_of_range 异常
  - 由于 size_t 是无符号整型，所以当传入负数时，被当作一个很大的正整数，往往造成数组越界

```
void f() {
    string s("hello world!");
    string s2(s, -3, 4); // -3转换成一个大正整数，越界，抛出
out_of_range
}
```

字符串 - 关于字符串类

- 转换到 C 风格的字符串

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    // ...
    size_type copy(CharT* s, size_type n, size_type pos = 0)
const;
    const CharT* c_str() const; // 返回指向带'\0'结尾的字符数组
    const CharT* data() const; // 注意：返回指向不带'\0'结尾的字符数组
的指针
    //...
};
```

注意：

1，c_str() 和 data() 都是将 string 中的字符写入数组，返回指向该数组的指针，但数组仍归该 string 对象拥有，在 string 内容发生改变后，该数组将失效！

2，如果要将 string 对象的字符数组复制出来，可以通过 copy() 函数，注意 copy() 函数不会在数组尾部加 '\0'

字符串 - 常用操作

- 比较 (comparisons)

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    // ...
    int compare(const basic_string& str) const;
    int compare(size_type pos, size_type n,
            const basic_string& str) const;
    int compare(size_type pos1, size_type n1,
            const basic_string& str, size_type pos2,
            size_type n2) const;
    int compare(const CharT* s) const;
    int compare(size_type pos, size_type n1, const CharT* s)
const;
    int compare(size_type pos, size_type n1, const CharT* s,
            size_type n2) const;
    //...
};
```

字符串 - 常用操作

- 插入 (Insert)

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    // ...
    basic_string& operator+=(const basic_string& str);
    basic_string& operator+=(const CharT* s);
    basic_string& operator+=(CharT c);

    basic_string& append(const basic_string& str);
    basic_string& append(const basic_string& str, size_type
pos, size_type n);
    basic_string& append(const CharT* s, size_type n);
    basic_string& append(const CharT* s);
    basic_string& append(size_type n, CharT c);
    template<class InputIterator>
    basic_string& append(InputIterator first, InputIterator
last);
    void push_back(CharT c);
```

字符串 - 常用操作

- 插入 (Insert)（续）

```
    void insert(iterator p, size_type n, CharT c);
    template<class InputIterator>
    void insert(iterator p, InputIterator beg, InputIterator
end);
    basic_string& insert(size_type pos1, const basic_string&
str);
    basic_string& insert(size_type pos1, const basic_string&
str, size_type pos2, size_type n);
    basic_string& insert(size_type pos, const CharT* s,
size_type n);
    basic_string& insert(size_type pos, const CharT* s);
    basic_string& insert(size_type pos, size_type n, CharT c);
    iterator insert(iterator p, CharT c);
    //...
};
```

字符串 - 常用操作

- 拼接 (Concatenation)
  - 通过操作符 + 进行拼接

字符串 - 常用操作

- 查找 (Find)

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    // ...
    size_type find(const CharT* s, size_type pos, size_type n)
const;
    size_type find(const basic_string& str, size_type pos = 0)
const;
    size_type find(const CharT* s, size_type pos = 0) const;
    size_type find(CharT c, size_type pos = 0) const;

    size_type rfind(const basic_string& str, size_type pos =
npos) const;
    size_type rfind(const CharT* s, size_type pos, size_type
n) const;
    size_type rfind(const CharT* s, size_type pos = npos)
const;
    size_type rfind(CharT c, size_type pos = npos) const;
```

字符串 - 常用操作

- 查找 (Find) （续 1 ）

```
    size_type find_first_of(const basic_string& str, size_type
pos = 0) const;
    size_type find_first_of(const CharT* s, size_type pos,
size_type n) const;
    size_type find_first_of(const CharT* s, size_type pos = 0)
const;
    size_type find_first_of(CharT c, size_type pos = 0) const;
    size_type find_last_of(const basic_string& str, size_type
pos = npos) const;
    size_type find_last_of(const CharT* s, size_type pos,
size_type n) const;

    size_type find_last_of(const CharT* s, size_type pos =
npos) const;
    size_type find_last_of(CharT c, size_type pos = npos)
const;
```

字符串 - 常用操作

- 查找 (Find) （续 2 ）

```
    size_type find_first_not_of(const basic_string& str,
size_type pos = 0) const;
    size_type find_first_not_of(const CharT* s, size_type pos,
            size_type n) const;
    size_type find_first_not_of(const CharT* s, size_type pos =
0) const;
    size_type find_first_not_of(CharT c, size_type pos = 0)
const;

    size_type find_last_not_of(const basic_string& str,
size_type pos = npos) const;
    size_type find_last_not_of(const CharT* s, size_type pos,
            size_type n) const;
    size_type find_last_not_of(const CharT* s, size_type pos =
npos) const;
    size_type find_last_not_of(CharT c, size_type pos = npos)
const;
    //...
};
```

字符串 - 常用操作

- 替换 (Replace)

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    basic_string& replace(size_type pos, size_type n, const
basic_string& str);
    basic_string& replace(size_type pos1, size_type n1,
       const basic_string& str, size_type pos2, size_type n2);
    basic_string& replace(size_type pos, size_type n1, const
CharT* s, size_type n2);
    basic_string& replace(size_type pos, size_type n1, const
CharT* s);
    basic_string& replace(size_type pos, size_type n1,
size_type n2, CharT c);
    basic_string& replace(iterator i1, iterator i2, const
basic_string& str);
    basic_string& replace(iterator i1, iterator i2, const
CharT* s, size_type n);
    basic_string& replace(iterator i1, iterator i2, const
CharT* s);
    basic_string& replace(iterator i1, iterator i2, size_type
n, CharT c);
```

字符串 - 常用操作

- 替换 (Replace)（续）

```
    template<class InputIterator>
    basic_string& replace(iterator i1, iterator i2,
InputIterator k1, InputIterator k2);
    basic_string& replace(iterator i1, iterator i2, CharT* k1,
CharT* k2);
    basic_string& replace(iterator i1, iterator i2, const
CharT* k1, const CharT* k2);
    basic_string& replace(iterator i1, iterator i2, iterator
k1, iterator k2);
    basic_string& replace(iterator i1, iterator i2,
const_iterator k1, const_iterator k2);

    basic_string& erase(size_type pos = 0, size_type n =
npos);
    iterator erase(iterator position);
    iterator erase(iterator first, iterator last);
    void clear();
    //...
};
```

字符串 - 常用操作

- 子串 (Substrings)
  - 从 string 中取字串，可以通过成员函数 substr()

```
basic_string
substr(size_type pos = 0, size_type n = npos) const
```

- 交换 (Swap)
  - 交换 2 个 string 对象的内容，使用 string 的 swap() 成员函数比算法库的 swap() 函数高效

```
void swap(basic_string& s);
```

字符串 - 常用操作

- 大小和容量 (Size and Capacity)

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string {
    // ...
    size_type size() const;
    size_type length() const;
    bool empty() const;

    size_type max_size() const;
    void resize(size_type n, CharT c);
    void resize(size_type n);
    size_type capacity() const;
    void reserve(size_type res_arg = 0);
    //...
};
```

字符串 - C 标准库字符串函数

- 常用字符串操作函数

```
// in <string.h>
char* strcpy (char* dest, const char* src);
/* Copy no more than N characters of SRC to DEST.  */
char* strncpy (char* dest, const char* src, size_t n);
/* Append SRC onto DEST.  */
char* strcat (char* dest, const char* src);
/* Append no more than N characters from SRC onto DEST.  */
char* strncat (char* dest, const char* src, size_t n);

/* Compare S1 and S2.  */
int strcmp (const char* s1, const char* s2);
/* Compare N characters of S1 and S2.  */
int strncmp (const char* s1, const char* s2, size_t n);

char* strchr (char* s, int c);
const char* strchr (const char* s, int c);
char* strrchr (char* s, int c);
const char* strrchr (const char* s, int c);
```

字符串 - C 标准库字符串函数

- 常用字符串操作函数（续）

```
char* strstr (char* haystack, const char* needle);
const char* strstr (const char* haystack, const char* needle);
char* strpbrk (char* s, const char* accept);
const char* strpbrk (const char* s, const char* accept);

size_t strcspn (const char* s, const char* reject);
/* Return the length of the initial segment of S which
   consists entirely of characters in ACCEPT.  */
size_t strspn (const char* s, const char* accept);

size_t strlen (const char* s);

/* Divide S into tokens separated by characters in DELIM.  */
char* strtok (char* s, const char* delim);
```

字符串 - C 标准库字符串函数

- 字符串转换为数值

```
// in <stdlib.h>
/* Convert a string to an integer.  */
int atoi (const char* nptr);
/* Convert a string to a long integer.  */
long int atol (const char* nptr);
/* Convert a string to a long integer.  */
long int strtol (const char* nptr, char** endptr, int base);

/* Convert a string to a floating-point number.  */
double atof (const char* nptr);
/* Convert a string to a floating-point number.  */
double strtod (const char* nptr, char** endptr);
```

字符串 - C 标准库字符串函数

- 字符类别 (Character Classification)

```
// in <ctype.h>
int isalnum(int); //[a-zA-Z0-9]
int isalpha(int); //[a-zA-Z]
int iscntrl(int); //ASCII (0 to 31, 127)
int isdigit(int); //[0-9]
int islower(int); //[a-z]
int isgraph(int); //ispunct|isalnum
int isprint(int); //可打印字符, ASCII (' ' to '~')
int ispunct(int); //标点符号
int isspace(int); //空白字符
int isupper(int); //[A-Z]
int isxdigit(int); //[0-9a-fA-F]

// 操作
int toupper(int);
int tolower(int);
```

字符串 - Bjarne's Advices

- Bjarne's Advices
  - 尽量用 string 操作，少用 C 风格的字符串函数
  - 用 string 作为变量或成员，不要作为基类
  - string 可以作为参数或返回值，不必担心存储管理问题
  - 如果希望做范围检查，用 at() 而不是下标或迭代器
  - 如果优化速度时，用 [] 而不是 at()
  - 直接或间接使用 substr() 读子串， replace() 写子串
  - 使用 string::npos 表示 " 字符串剩余的部分 "
  - 注意不要将带 0 字符串的 char* 传给字符串函数
  - 只有在必要的情况下，才将 string 内容用 c_str() 转为 C 字符串
  - 使用字符类别操作 (isalpha 等)，而不是检测字符的数值