String复习

1. 标准库中的string类

1.1 string类

string类标准库

1.2 auto和范围for

auto关键字

- 在早期C/C++中auto的含义是：使用**auto修饰的变量，是具有自动存储器的局部变量，后来这个不重要了。**C++11中，标准委员会变废为宝赋予了auto全新的含义即：**auto不再是一个存储类型 指示符，而是作为一个新的类型指示符来指示编译器，auto声明的变量必须由编译器在编译时期推导而得。**
- **用auto声明指针类型时，用auto和auto*没有任何区别，但用auto声明引用类型时则必须加&**
- **当在同一行声明多个变量时，这些变量必须是相同的类型，否则编译器将会报错，因为编译器实际只对第一个类型进行推导，然后用推导出来的类型定义其他变量。**
- **auto不能作为函数的参数，可以做返回值，但是建议谨慎使用**
- **auto不能直接用来声明数组**

代码块

```
#include<iostream>
#include <string>
#include <map>

using namespace std;

int main()
{
    std::map<std::string, std::string> dict = { { "apple", "苹果" },{ "orange",
    "橙子" }, {"pear","梨"} };
    // auto的用武之地
    //std::map<std::string, std::string>::iterator it = dict.begin();
    auto it = dict.begin();
    while (it != dict.end())
    {
```

```
        cout << it->first << ":" << it->second << endl;
        ++it;
    }

    return 0;
}

```

范围for

- 对于一个**有范围的集合**而言，由程序员来说明循环的范围是多余的，有时候还会容易犯错误。因此C++11中引入了基于范围的for循环。**for循环后的括号由冒号“：”分为两部分：第一部分是范围内用于迭代的变量，第二部分则表示被迭代的范围**，自动迭代，自动取数据，自动判断结束。
- 范围for可以作用到数组和容器对象上进行遍历
- 范围for的底层很简单，容器遍历实际就是替换为迭代器，这个从汇编层也可以看到。

代码块

```
#include<iostream>
#include <string>
#include <map>
using namespace std;
int main()
{
    int array[] = { 1, 2, 3, 4, 5 };
    // C++98的遍历
    for (int i = 0; i < sizeof(array) / sizeof(array[0]); ++i)
    {
        array[i] *= 2;
    }
    for (int i = 0; i < sizeof(array) / sizeof(array[0]); ++i)
    {
        cout << array[i] << endl;
    }

    // C++11的遍历
    for (auto& e : array) e *= 2;

    for (auto e : array) cout << e << " " << endl;

    string str("hello world");
    for (auto ch : str)
    {
        cout << ch << " ";
```

```
27        }
28        cout << endl;
29
30        return 0;
31    }
```

1.3 string常用接口

1.3.1 string类常用构造

(constructor)函数名称	功能说明
string() （重点）	构造空的string类对象，即空字符串
string(const char* s) （重点）	用C-string来构造string类对象
string(size_t n, char c)	string类对象中包含n个字符c
string(const string&s) （重点）	拷贝构造函数

1.3.2 string类对象的容量操作

函数名称	功能说明
size（重点）	返回字符串有效字符长度
length	返回字符串有效字符长度
capacity	返回空间大小
empty （重点）	检测字符串释放为空串，是返回true，否则返回false
clear （重点）	清空有效字符
reserve （重点）	为字符串预留空间
resize （重点）	将有效字符的个数改成n个，多出的空间用字符c填充

> 注意：

1. size()与length()方法底层实现原理完全相同，引入size()的原因是为了与其他容器的接口保持一致，一般情况下基本都是用size()。
2. clear()只是将string中有效字符清空，不改变底层空间大小。
3. resize(size_t n) 与 resize(size_t n, char c)都是将字符串中有效字符个数改变到n个，不同的是当字符个数增多时：resize(n)用0来填充多出的元素空间，resize(size_t n, char c)用字符c来填充多出的元素空间。注意：resize在改变元素个数时，如果是将元素个数增多，可能会改变底层容量的大小，如果是将元素个数减少，底层空间总大小不变。
4. reserve(size_t res_arg=0)：为string预留空间，不改变有效元素个数，当reserve的参数小于string的底层空间总大小时，reserver不会改变容量大小。

1.3.3 string类对象的访问及遍历操作

函数名称	功能说明
operator[] （重点）	返回pos位置的字符，const string类对象调用
begin+ end	begin获取一个字符的迭代器 + end获取最后一个字符下一个位置的迭代器
rbegin + rend	begin获取一个字符的迭代器 + end获取最后一个字符下一个位置的迭代器
范围for	C++11支持更简洁的范围for的新遍历方式

1.3.4 string类对象的修改操作

函数名称	功能说明
push_back	在字符串后尾插字符c
append	在字符串后追加一个字符串
operator+= (重点)	在字符串后追加字符串str
c_str(重点)	返回C格式字符串
find + npos(重点)	从字符串pos位置开始往后找字符c，返回该字符在字符串中的位置

rfind	**从字符串pos位置开始往前找字符c，返回该字符在字符串中的位置**
substr	**在str中从pos位置开始，截取n个字符，然后将其返回**

1.3.5 string类非成员函数

函数	功能说明
operator+	尽量少用，因为传值返回，导致深拷贝效率低
operator>> （重点）	输入运算符重载
operator<< （重点）	输出运算符重载
getline （重点）	获取一行字符串
relational operators （重点）	大小比较

2. 自主实现的MyString

实现代码

代码块

```
#define _CRT_SECURE_NO_WARNINGS
#pragma once

#include<iostream>

namespace String
{
    class string
    {
    public:
        typedef char* iterator;
        typedef const char* const_iterator;

        iterator begin() { return _str; }
        iterator end() { return _str + _size; }
        const_iterator begin() const { return _str; }
        const_iterator end() const { return _str + _size; }

        string(const char* str = "")
```

```
20          {
21              _size = strlen(str);
22              _capacity = _size;
23
24              _str = new char[_capacity + 1];
25
26              for (int i = 0; i < _size; i++) _str[i] = str[i];
27              _str[_size] = '\0';
28          }
29
30          void swap(string& s)
31          {
32              _size = s.size();
33              _capacity = s._capacity;
34              for (int i = 0; i <= _size; i++) _str[i] = s[i];
35          }
36
37          string(const string& s)
38          {
39              string tmp = string(s);
40              swap(tmp);
41          }
42
43          string& operator=(string& s)
44          {
45              swap(s);
46              return *this;
47          }
48
49          const char* c_str() const { return _str; }
50
51          void clear() // str清空
52          {
53              _size = 0;
54              _str[0] = '\0';
55          }
56
57          size_t size() const { return _size; }
58          size_t capacity() const { return _capacity; }
59
60          char& operator[](size_t pos) { return _str[pos]; }
61          const char& operator[](size_t pos) const { return _str[pos]; }
62
63          void reserve(size_t n)
64          {
65              if (n > _capacity)
66              {
```

```
                char* tmp = new char[n + 1];
                strcpy(tmp, _str);
                delete[] _str;
                _str = tmp;
                _capacity = n;
            }
        }

        void resize(size_t n)
        {
            _size = n;
            _str[_size] = '\0';
        }

        void resize(size_t n, char c)
        {
            _size = n;
            _str[_size] = '\0';

            for (int i = 0; i < _size; i++) _str[i] = c;
        }

```

代码块

```
        void push_back(char ch)
        {
            _capacity == 0 ? 4 : _capacity;
            if (_size + 1 > _capacity) _capacity *= 2;

            _str[_size] = ch;
            _size++;
            _str[_size] = '\0';
        }

        void append(const char* str)
        {
            int len = strlen(str);
            if (_size + len > _capacity)
            {
                reserve(len + _size > 2 * _capacity ? len + _size : 2 *
    _capacity);
            }
            strcpy(_str + size(), str);
            _size += len;
```

```
20          }
21
22          string& operator+=(char ch)
23          {
24              push_back(ch);
25              return *this;
26          }
27
28          string& operator+=(const char* str)
29          {
30              append(str);
31              return *this;
32          }
33
34          string& insert(size_t pos, const string& str)
35          {
36              size_t size = str.size();
37              if (_size + size > _capacity && _size + size < _capacity * 2)
    _capacity *= 2;
38              else if (_size + size > _capacity * 2) reserve(_size + size);
39
40              for (int i = _size; i > pos; i--)
41                  _str[i + size] = _str[i];
42
43              for (int i = 0; i < size; i++)
44                  _str[i + pos] = str[i];
45
46              _size += size;
47              return *this;
48          }
49
50          string& insert(size_t pos, const char* str)
51          {
52              string tmp(str);
53              insert(pos, tmp);
54              return *this;
55          }
56
57          string& insert(size_t pos, const char* str, size_t n)
58          {
59              char* tmp = new char[n + 1];
60              strncpy(tmp, str, n);
61              tmp[n] = '\0';
62
63              insert(pos, tmp);
64              return *this;
65          }
```

```
66
67          void erase(size_t pos = 0, size_t len = npos)
68          {
69              if (len == npos || len > _size || len <= 0) len = _size - pos;
70              if (len == _size) clear();
71
72              for (int i = pos + len; i < _size; i++)
73                      _str[i - len] = _str[i];
74
75              _size -= len;
76              _str[_size] = '\0';
77          }
78
79
80          size_t find(char ch, size_t pos = 0)
81          {
82              for (int i = pos; i < _size; i++)
83                  if (_str[i] == ch) return i;
84
85              return npos;
```

代码块

```
1   size_t find(const char* str, size_t pos = 0)
2           {
3               size_t size = strlen(str);
4               for (int i = pos; i <= _size - size; i++)
5                   if (_str[i] == str[0] && strcmp(substr(i, size).c_str(), str)
    == 0) return i;
6
7               return npos;
8           }
9
10          string& substr(size_t pos = 0, size_t len = npos)
11          {
12              len = len == npos ? _size - pos : len;
13              char* tmp = new char[len + 1];
14
15              for (int i = 0; i < len; i++) tmp[i] = _str[i + pos];
16
17              tmp[len] = '\0';
18              string s(tmp);
19              return s;
20          }
21
```

```
22          ~string() {}
23
24      private:
25          char* _str;
26          size_t _size;
27          size_t _capacity;
28
29          static const int npos = -1;
30      };
31
32      bool operator <(const string& s1, const string& s2) { return
    strcmp(s1.c_str(), s2.c_str()) < 0; }
33
34      bool operator ==(const string& s1, const string& s2) { return
    strcmp(s1.c_str(), s2.c_str()) == 0; }
35
36      bool operator <=(const string& s1, const string& s2) { return (s1 < s2) ||
    (s1 == s2); }
37
38      bool operator >(const string& s1, const string& s2) { return !(s1 <= s2); }
39
40      bool operator >=(const string& s1, const string& s2) { return !(s1 < s2); }
41
42      bool operator !=(const string& s1, const string& s2) { return !(s1 == s2);
    }
43
44      std::ostream& operator <<(std::ostream& out, const string& str)
45      {
46          for (auto ch : str) out << ch;
47          return out;
48      }
49
50      std::istream& operator >>(std::istream& in, string& str)
51      {
52          const int N = 256;
53          char buffer[256];
54
55          size_t index = 0;
56          char ch = in.get();
57          while (ch != ' ' && ch != '\n')
58          {
59                  buffer[index++] = ch;
60                  ch = in.get();
61
62                  if (index == N - 1)
63                  {
64                          buffer[index] = '\0';
```

```
65                          str += buffer;
66                          index = 0;
67                  }
68          }
69
70          if (index > 0)
71          {
72                  buffer[index] = '\0';
73                  str += buffer;
74          }
75
76          return in;
77      }
78  }
```