C风格字符串

劣势

- 字符数组，以终止符 `\0` 结尾，内存管理麻烦
- 不支持 `=/==/</>/<=/>=/` 等运算符
- C风格字符串不能作为map和set类型参数
- 功能少，无法提供取子串、匹配等功能

优势

可以直接使用 `printf` 和 `scanf`

C++字符串

STL库的字符串使用 `#include <string>` 导入

- 像内置类型（int/long/double/float），支持 `=/==/</>/<=/>=/` 等运算符
- 类似于 `vector<char>`，支持 `push_back/insert/迭代器`
- 额外的功能，如求子串、匹配

> ⚠ **Warning**
>
> string不能 `printf` 和 `scanf`

基本操作

赋值

```
#include <string>
string str = "hello"; // string 支持=赋值
string str1 = str;
```

判断内容是否相同

```
bool isSame = (str == str1);
```

加法

```
str3 = str + str1; // 连接操作
```

比较大小

比较大小比较的是字典序

```
// string支持< <= > >=比较大小
bool isLarge = (str1 > str);
```

vector<char>功能

string非常像vector<char>。

访问元素

```
string str4 = "abcdef";
char ch;
ch = str4[0];
```

push_back/pop_back

```
str4.push_back('g');
str4.pop_back();
```

迭代器

```
string:iterator it;
for(it = str.begin(); it != str.end(); it++) {
    printf("%c\n", *it);
}
```

插入和删除

```
it = str4.begin();
str4.insert(it, 'A'); // 插入
it = str4.end();
str4.erase(it); // 删除
```

额外功能

拓展insert和erase用法

- 使用整数下标可以一次插入多个字符

```
str4.insert(0, "xyz"); // 第一个参数是一个整数下标，第二个参数是字符串常量
```

- 使用整数下标可以一次删除多个字符

```
str4.erase(0, 3); // 第一个参数是开始删除的下标，第二个参数是删除的个数
```

> **Caution**
>
> 删除的字符不包括第二个参数位置的字符

获取子串

```
string str5 = str4.substr(0, 3); // 第一个参数是起点下标，第二个参数是子串长度
```

字符串匹配

```
string str6 = "howdoyoudo";
int pos = str6.find("do", 0); // 第二个参数是开始查找的下标，如果找不到返回-1
if(pos == string::npos) { // string::npos就是-1，一般使用这种写法
    printf("do is not found!\n");
}
```

string和数值相互转化

- 数值转化为字符串

  `to_string` 方法

- 字符串转化为数值

  使用 `sto` 系列函数，如 `stoi` 转化为int类型， `stol` 转化为long类型， `stof` 转化为float类型， `stod` 转化为double类型

```
/*数值转化为字符串*/
int i = 1234;
string str7 = to_string(i);
float f = 3.14;
str7 = to_string(f);

/*字符串转化为数值*/
string str8 = "3.14159";
f = stof(str8);
str8 = "314159";
i = stoi(str8);
```

输入和输出

方案一：继续用 `scanf` 和 `printf`

```
/*输入：先读到字符数组，再转成字符串*/
char arr[100];
scanf("%s", arr);
string str9 = arr;

/*输出：使用c_str函数转成字符数组*/
printf("%s\n", str9.c_str());
```

方案二（不推荐）：用c++的 `cin` 和 `cout`

需要包含头文件 `#include <iostream>`

```cpp
#include <iostream>

string str10;
cin >> str10; // 输入
cout << "str10 = " << str10 << "\n";  // 输出
```

> **Caution**
>
> - 性能差，非常容易超时
> - 格式管理麻烦

读取一整行字符串

`scanf`和`cin`默认只能读取空格前面的字符串，不能连同空格一起读入。

方式一：fgets

```cpp
char arr[200] = { 0 };
fgets(arr, 200, stdin); // 第一个参数填写字符数组起始地址，第二个参数填写字符数组的长度，第三个参数填写stdin
```

> **Warning**
>
> 会连同换行符`\n`一起读入字符数组，可以把换行符作为字符串边界
>
> ```cpp
> while (true) {
>  if (arr[i] == '\0' || arr[i] == '\n') {
>      break; // 已经到达字符串末尾
>  }
>  ......
> }
> ```

方式二：cin.getline