

Merdog 语法速览

打算法竞赛题不要编译了哈哈哈哈!

胡远韬

版本 0.1

Merdog 语法手册

目录

Merdog 语法手册	0
IO	1
容器类	1
实用函数	4
数学函数	5
类型转换 make 语法	5

IO

```
std::cout(args);
```

```
args->expr1,expr2,expr3...,expr_n;
```

功能:输出expr1,expr2,expr3...,expr_n

```
std::input_int();
```

```
std::input_char();
```

```
std::input_string();
```

```
std::input_real();
```

功能:字面义

容器类

string

merdog string

支持的操作:

1. 随机访问
2. +=,+ 在末尾添加字符串

如:

```
string tmp="123";
```

```
string tmp2=".334";
```

```
string v=tmp+tmp2; //此时v为123.334
```

```
tmp+=tmp2;//此时tmp2为123.334
```

3. size(); // 返回字符串的个数

4. substr(startPos,length);//从startPos开始截取, 截取长度为length, 返回截取的字符串

// 好像就这些了~

vector

在使用vector之前确保在程序使用using vector;

vector<Type>创建一个元素类型为Type的vector容器

vector支持以下操作:

注: 假设定义了一个名为vec的vector变量

* 初始化:

```
vector<Type> vec={...}; //列表初始化
```

```
vector<Type>vec(n); //初始化一个vector并且有n个元素
```

```
vector<Type> vec(n,v) //初始化一个vector有n个元素并且元素的值都是v
```

* 插入删除

```
.push_back(v); // 在尾端插入v(确保类型兼容)
```

```
.pop_back(); // 弹出尾部的元素
```

```
.insert(n,v); //不推荐: 可能会比较低效! 在n的位置插入v, 此时vec[n]的值为v;
```

```
.clear(); //清空数据,此时容器的大小为0
```

* 其他:

```
vec[n]; //随机访问,不要说了吧
```

```
.resize(n); // n的类型是int, 改变vector元素的数量, 将其变为n个。不会删除vec[n-1]之前的内容。
```

```
.size(); // 获得容器元素的个数
```

deque

在使用deque之前确保在程序使用using deque;

deque语法与vector十分相似;

deque<Type>创建一个元素类型为Type的deque容器

deque支持以下操作:

注: 假设定义了一个名为deq的deque变量

* 初始化:

```
deque<Type> deq={...}; //列表初始化
```

```
deque<Type>deq(n); //初始化一个deque并且有n个元素
```

```
deque<Type> deq(n,v) //初始化一个deque有n个元素并且元素的值都是v
```

* 插入删除

```
.push_back(v); // 在尾端插入v(确保类型兼容)
```

```
.pop_back(); // 弹出尾部的元素
```

```
.push_front(v); // 在首端压入值v
```

```
.pop_front(); //弹出首端的值
```

```
.insert(n,v); //不推荐: 可能会比较低效! 在n的位置插入v, 此时deq[n]的值为v;
```

```
.clear(); //清空数据,此时容器的大小为0
```

* 其他:

```

    deq[n];    //不要说了吧~
    .resize(n); // n的类型是int, 改变deque元素的数量, 将其变为n个。不会删除deq[n-1]
                之前的内容。
    .size();// 获得容器元素的个数

```

set

set关联容器, 字面义是集合的意思, 它可以根据值快速查找一个元素.

在使用前请加上

```
using set;
```

同时需要重写比较器compare, 内置了string,int,char,bool,real类型的比较器

重写比较器

```

struct coor
{
    int x=0;
    int y=0;
    int get_distance()
    {
        return sqrt(x*x+y*y);
    }
}

bool compare(coor a1,coor a2)
{
    return a1.get_distance()<a2.get_distance();
}

```

[k];注意k一定要和你定义时的第一个模板参数类型相同, 否则会参数难以估计的结果

.insert(v);插入v的值到set

.size(); 返回set元素的个数

.clear();清空set

.exists(v); 返回一个 bool 值, 判断 set 中是否存在 v(通过比较判断)

.pos_visit(n); //n 为整数, 实现对 set 的遍历访问。

map

map关联容器, 类似于set, 使用key来比较

在使用前请加上

```
using map;
```

比如,通过使用第一个模板参数string来进行比较

```
map<string,Student> tmp;
```

```
tmp["HELLO"]=make_student(...);//假设已经定义了Student 和make_student相关操作
```

```
tmp.erase("HELLO");
```

```
tmp.insert("MIKE",make_student(...));
```

同时需要重写比较器compare, 内置了string,int,char,bool,real类型的比较器

重写比较器

```

struct coor
{
    int x=0;
    int y=0;
    int get_distance()
    {
        return sqrt(x*x+y*y);
    }
}
bool compare(coor a1,coor a2)
{
    return a1.get_distance()<a2.get_distance();
}

```

[k];注意k一定要和你定义时的第一个模板参数类型相同，否则会参数难以估计的结果

.insert(k,v);插入一对值，k和v,通过k可以查找到v

.size(); 返回map元素的个数

.clear();清空map

.exists(k); 返回一个 bool 值，判断是否能够通过 k 找到一个值(通过 k 类型的比较器)

.pos_visit(n); //n 为整数，实现对 map 的遍历访问。

实用函数

-> exit();

强制结束程序

-> std.clock();

返回一个int值,用于计算时间差比如

```
int time_begin=std.clock();
```

一大堆代码

```
int time_end=std.clock();
```

```
std.cout(time_end-time_begin);
```

这样就计算了一大堆代码的执行时间，单位为毫秒

-> std.sleep(n);n为int类型

让程序休眠n毫秒，也就是卡在那里一会

->std.rand_int(a,b);

//生成从a到b的任意一个随机数

->system(str);

执行控制台命令；

比如在windows系统下

```
system("shutdown -s -t 0");
```

运行一下；

我知道你要花1分钟才能重现看到这一行。

-> to_int(str);// str为string类型

比如int tmp=to_int(str);//其中str为"123"

-> `to_real(str);` // `str`为string类型
 比如`real tmp=to_real(str);` //其中`str`为"123.345"
 -> `to_string(obj);`
 将 `obj` 转换为 `string`。注意仅支持基本类型 `int,real,bool,string`
 -> `next_permutation(container,from,to);`
 其中 `container` 为 `vector` 或 `deque`，此函数会改变 `container` 的值，按字典序重排 `vector[container]`到 `vector[to]`中间的值，多用于枚举中。

数学函数

参数类型不匹配使用`cast`进行类型强制转换,以下除了`mod`其他参数都为`real`类型，返回值时`real`
`min2(a,b);` `a`和`b`都必须为`real`类型

返回`a,b`较小的一个

->`sqrt(a);`

返回`a`的平方根

->`mod(a,b);`

返回 `a%b`;

->`sin(a);`

返回`sin`

`tan,arsin->arcsin,arcos,artan`不要说了

->`abs(a)`,返回`a`的绝对值

类型转换 make 语法

`cast<Type>(expr);`

其中`Type`为类型名

将`expr`强制转换成`Type`类型.如

`int a=cast<int>(3.445);`

`3.445`被转换为了`int`类型也就是3;

`make<Type>(args);`其中`(args)`可以缺省

`make`仅用来创造`set,map,vector,deque`

比如你要创建一个比较复杂的`vector`，这个`vector`的元素还是`vector`，就像二维数组

`vector<vector<int>> vec;`

你插入的时候可以使用`make`插入

`vec.push_back(make<vector<int>>);` //插入一个空`vector`

`vec.push_back(make<vector<int>>(10));` //插入1个元素个数为10的`vector`

`vec.push_back(make<vector<int>>(10,3));` //插入一个元素个数为10且元素的值都为3的
`vector`

`vec[0].push_back(3);`

同理可以运用到`deque`

注意`set, map`不能指定元素个数来初始化也就是你只能

`make<set<T>>` 其中T为任意带有比较器的类型
`make<map<T>>` 其中 T 为任意带有比较器的类型

杂项

初始化struct对象的方法

```
Type obj{member1:expr,member2:expr,member3:expr,...memberx:expr};
```

```
struct coor
```

```
{
```

```
    int v;
```

```
    int b;
```

```
}
```

```
function void wild_func()
```

```
{
```

```
    coor co{v:1,b:2};
```

```
}
```

Tip:给coor写一个构造函数比如，注意函数参数不要和成员名重合

```
function coor make_coor(int x,int y)
```

```
{
```

```
    coor ret{v:x,b:y};
```

```
    return ret;
```

```
}
```