ACM Data Maker

@KeadinZhou

一、准备工作

在开始前，请确保 `g++` 和 `python` (*python3*)已经正确的配置到系统路径中。

当前仅支持Windows系统

二、导入Maker

下载并在你的代码中引入 `ACM_DATA_MAKER.H`。

```
#include "ACM_DATA_MAKER.H"
```

三、环境初始化

```
makerInit();
```

初始化函数将会创建一个名为 `_data` 的临时目录，并将答案生成脚本写入到该目录，当前目录中的标程文件`std.cpp`也将会被拷贝到这个目录中（在此之前，请确认标程文件名为`std.cpp`，且在当前目录中——当然，在数据生成完成后自行拷贝到临时目录也没有问题）。

生成的 `__makeout.bat` 和 `__win2nix.py` 会在之后的答案生成时用到，前者的作用是生成答案文件，后者的作用是删除所有数据文件中存在的 `\r`。（由于的系统权限不同，程序将不会自动运行 `__makeout.bat`，需要自己手动运行）。

四、函数列表

1. 输出

write()

```
template<typename T, typename... REST> inline void write(T x, REST... rest);
template<typename T> inline void write(T x);
```

输出，支持不定长参数表，多个参数时，每个参数将会用空格隔开（最后一个参数后面没有空格也没有回车）。

特殊的，`write()`函数支持`vector<>`。

```
write(1); //1
write(1,2,3); //1 2 3
vector<int> ve={1,2,3};
write(ve); //1 2 3
```

writeln()

```
template<typename... REST> inline void writeln(REST... rest);
```

输出后换行。在`write()`的基础上，末尾增加一个换行。

changeVectorSpacing()

```
void changeVectorSpacing(string s);
```

更改`vector<>`输出时，各个元素之间的分隔符（默认为一个空格）。

```
vector<int> ve={1,2,3};
write(ve); //1 2 3
changeVectorSpacing(",");
write(ve); //1,2,3
```

newFile()

```
int newFile();
int newFile(int index);
```

生成一个新的`*.in`数据文件，不带参数默认从`1`开始增长，带参数指定文件名。一般由`*DataBuilder`自动调用，无需自行调用，运行成功后回返回当前的文件序号。

backToTerminal()

```
void backToTerminal();
```

将标准输出流重新指回控制台。

注意：输出的调试信息之前就应该调用该函数，否则调试信息将被写入数据文件。

2. 数据生成

randomInt()

```
int randomInt();
int randomInt(int MIN,int MAX);
```

生成一个`[0,32768)`范围内的随机整数，带两个参数时，指定生成的范围（`MIN<=MAX`，当指定的范围超出时，会被`assert`掉）

推荐使用`randomBigInt()`而不是该函数

```
writeln(randomInt()); //11138
writeln(randomInt(10,55)); //12
```

randomBigInt()

```
LL randomBigInt(LL MIN, LL MAX);
```

生成一个`[MIN,MAX]`范围内的整数，可到`long long`级别。

```
writeln(randomBigInt(1,1e18)); //688165822120877625
```

randomDouble()

```
double randomDouble(LL MIN, LL MAX);
```

生成一个 `[MIN,MAX]` 范围内的实数。

```
writeln(randomDouble(1,1e18)); //7.92893e+017
```

randomString()

```
string randomString(int len,string pool=ALPHA_STRING);
```

生成一个长度为 `len` 的随机字符串，所有字符从给定的 `pool` 串中随机选择，当不指定 `pool` 串时，默认为 `ALPHA_STRING`，即字母串。

系统预先定义了几个常量串，可以直接使用：

```
const string LOWERCASE_STRING = "abcdefghijklmnopqrstuvwxyz";   //小写字母
const string UPPERCASE_STRING = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";   //大写字母
const string DIGIT_STRING = "0123456789";   //数字
const string ALPHA_STRING = LOWERCASE_STRING + UPPERCASE_STRING;    //字母
```

当要求每个字符在结果中出现的概率不同时，可调整字符在 `pool` 出现的次数。

```
writeln(randomString(10));  //FjxsjttjHF
writeln(randomString(10,DIGIT_STRING)); //1820428606
writeln(randomString(10,"01")); //1011110001
```

randomVectorOfInt()

```
vector<int>randomVectorOfInt(int len,int MIN,int MAX,bool distinct=false);
```

生成一个有 `len` 个 `int` 的 `vector<>`，每个整数的范围为 `[MIN,MAX]`；当指定 `distinct` 为 `true` 时，生成的元素互不相同（不指定为 `false`），当范围不足以生成互不相同的元素时，会被 `assert` 掉。

```
writeln(randomVectorOfInt(10,0,15));    //14 4 8 14 10 4 5 4 1 7
writeln(randomVectorOfInt(10,0,15,true));   //6 15 3 5 1 4 8 11 9 2
```

randomVectorOfLL()

```
vector<LL>randomVectorOfLL(int len,LL MIN,LL MAX,bool distinct=false);
```

同 `randomVectorOfInt`，元素范围被扩大到 `long long` 范围。

randomVectorOfIntWithInit()

```
vector<int>randomVectorOfIntWithInit(int len,int MIN,int MAX,vector<int>
initVe,bool distinct=false);
```

同 `randomVectorOfInt`，接受一个 `initVe`，返回的数据中一定会包含其中的元素。

注意：当启用 `distinct` 时，函数并不会检查 `initVe` 是否合法。

randomVectorOfIntWithSumLimit()

```
vector<int>randomVectorOfIntWithSumLimit(int len,int MIN,int MAX,int sumLimit);
```

生成一个有 `len` 个 `int` 的 `vector<>`，每个整数的范围为 `[MIN,MAX]`，总和为 `sumLimit`，当无法满足条件时，会被 `assert` 掉。

```
writeln(randomVectorOfIntWithSumLimit(10,1,100,200));   //21 18 17 20 21 17 31
19 23 13
```

randomVectorOfLLWithSumLimit()

```
vector<LL>randomVectorOfLLWithSumLimit(int len,LL MIN,LL MAX,LL sumLimit);
```

同 `randomVectorOfIntWithSumLimit()`，范围扩展到 `long long`。

randomVectorOfString()

```
vector<string>randomVectorOfString(int len,int minStringLen,int
maxStringLen,string pool,bool distinct=false);
```

生成一个有 `len` 个 `string` 的 `vector<>`，每个字符串由 `pool` 串中生成，每个字符串的长度范围为 `[minStringLen,maxStringLen]`；当指定 `distinct` 为 `true` 时，生成的元素互不相同（不指定为 `false`）。

注意：启用`distinct`且不足以生成指定个数元素时，会导致死循环

```
writeln(randomVectorOfString(5,2,3,"01"));  //10 00 010 00 101
writeln(randomVectorOfString(5,2,3,"01",true)); //010 00 10 100 01
```

randomVectorOfStringWithSumLimit()

```
vector<string>randomVectorOfStringWithSumLimit(int len,int sumLimit,string
pool);
```

生成一个有 `len` 个 `string` 的 `vector<>`，每个字符串由 `pool` 串中生成，所有字符串的总长度为 `sumLimit`。

randomVectorOfStringWithSumLimitAndEachLimit()

```
vector<string>randomVectorOfStringWithSumLimitAndEachLimit(int len,int
minEach,int maxEach,int sumLimit,string pool);
```

生成一个有 `len` 个 `string` 的 `vector<>`，每个字符串由 `pool` 串中生成，每个字符串的长度在 `[minEach,maxEach]` 范围内，所有字符串的总长度为 `sumLimit`。

3.数据构造器

数据构造器的作用是进行更方便的数据文件管理，在数据生成的过程中，只需要关注每个数据之间的不同之处或是若干个特殊数据的生成，而不需要考虑数据的文件流向。**（当然，数据文件默认会按顺序写入工作目录的 `_data` 文件夹中，在开始之前，你得确保工作目录中不存在 `_data` 文件夹，或是在 `_data` 文件夹中没有重要的数据文件）**

（1）随机数据构造器

随机数据构造器适用于若干组只有数据规模有区别的数据。

我们分别用 `Targs` 和 `_Targs` 类型来描述一组数据的规模和若干组数据的规模，他们的实质都是 `vector<>` 和 `vector<>` 的嵌套，它们的原型如下：

```
typedef vector<int> Targs;  //一个数据点
typedef vector<Targs> _Targs;   //若干个数据点
```

可以看出，每项数据规模是用 `int` 类型来描述的。

随机数据构造器的构造器原型如下：

```
RandomDataBuilder(_Targs args,void(*func)(Targs));
```

第一个参数 `args` 是一个包含若干组数据规模信息的的列表，第二个参数 `func` 是一个函数原型为 `void func(Targs args)` 的函数指针。

构造完成后，随机数据构造器会将 `args` 中的每个元素传递给 `func` 函数，每个元素都会产生一个数据文件，你只需要编写一个通用的数据生成函数来接收`Targs` 类型的数据规模描述参数，并对不同的数据规模做出反应即可。

~~没有听明白的话请移步Demo，我发现我也讲不清楚~~

（2）特殊数据构造器

题目数据必然会包含经过构造的特殊数据点，这时候就需要用到特殊数据构造器了。

特殊数据构造器的构造器原型如下：

```
SpecialDataBuilder(vector<void(*)()> sp_list);
```

构造参数 `sp_list` 是包含 `void(*)()` 类型（无参数，无返回值）函数指针的`vector<>`。

构造完成后，特殊数据构造器会生成数据文件，并依次运行列表中的函数。

五、Demo

假设你现在要生成下面题目的数据文件：

> 第一行给一个整数 $N[0,100]$,
>
> 第二行给 N 个整数 $a_i[0,10^9]$,

> 求这 N 个整数的和。

你打算生成 6 个数据，其中 4 个数据是随机数据，这四个数据的数据规模如下：

- $N = 10, a_i \leq 10^2$
- $N = 50, a_i \leq 10^3$
- $N = 100, a_i \leq 10^6$
- $N = 100, a_i \leq 10^9$

以及两个特殊数据：

- $N = 10, a = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- $N = 0, a = \{\}$

那么对于随机数据部分，你只需要指定如下的参数列表：

```
_Targs _args={
    {10,(int)1e2},
    {50,(int)1e3},
    {100,(int)1e6},
    {100,(int)1e9},
};
```

以及编写一个通用的数据生成函数：

```
void randomData(Targs args){
    int N = args[0];        //接收第一个参数
    int maxAI = args[1];    //接收第二个参数

    writeln(N);     //输出第一行的n
    writeln(randomVectorOfInt(N,0,maxAI));      //随机成成一个vector<int>，并在一行
中输出
}
```

并用上面两个作为参数，生成一个 **随机数据构造器**：

```
new RandomDataBuilder(_args,randomData);
```

那么对于特殊数据部分，你只需要编写以下两个函数：

```
void sp1(){
    writeln(10);
    writeln(1,2,3,4,5,6,7,8,9,10);
}

void sp2(){
    writeln(0);
    writeln(vector<int>());
}
```

并将这两个函数封装成 `vector<>` 后传给 随机数据构造器的构造器即可。

```
new SpecialDataBuilder({sp1,sp2});
```

当然，在这此之前，需要初始化环境：

```
makerInit();
```

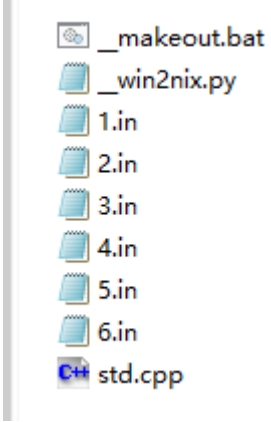
完整代码如下：

```
#include "ACM_DATA_MAKER.H"

_Targs _args={
    {10,(int)1e2},
    {50,(int)1e3},
    {100,(int)1e6},
    {100,(int)1e9},
};

void randomData(Targs args){
    int N = args[0];
    int maxAI = args[1];

    writeln(N);
    writeln(randomVectorOfInt(N,0,maxAI));
}

void sp1(){
    writeln(10);
    writeln(1,2,3,4,5,6,7,8,9,10);
}

void sp2(){
    writeln(0);
    writeln(vector<int>());
}

int main(){
    makerInit();
    new RandomDataBuilder(_args,randomData);
    new SpecialDataBuilder({sp1,sp2});
}
```

将标程重命名为 `std.cpp` 并放置在工作目录，这里给出一个当前题目的标程：

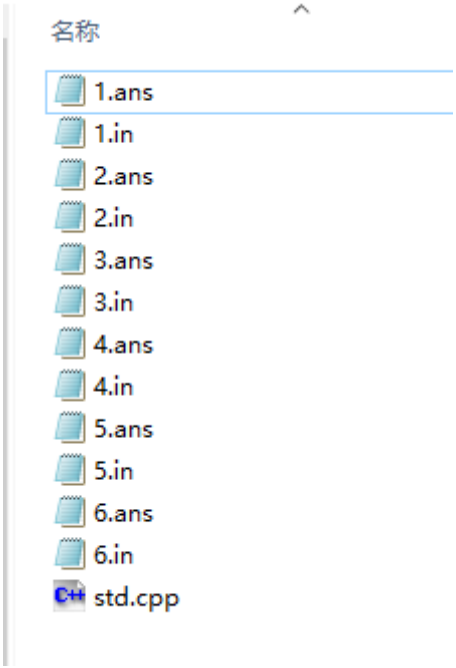
```
#include<bits/stdc++.h>
#define LL long long
using namespace std;

int main(){
    //Keadin

    int n;scanf("%d",&n);
    LL ans=0;
    while(n--){
        LL x;scanf("%lld",&x);
        ans+=x;
    }
    printf("%lld\n",ans);

    return 0;
}
```

运行之前数据生成代码，系统会生成一个 `_data` 目录：



如果没有 `std.cpp` 也可手动复制进去（*Clion会有奇怪的bug复制不成功*）。

此时只有 `*.in` 的输入数据文件，运行 `_makeout.bat`（*可能需要管理员权限*），即可生成所有对应的 `*.ans` 文件。



至此，所有数据都已经生成完成。

六、其它

目前的版本只能生成一些简单的数据，复杂的结构性数据（比如树、图之类的）目前仍需要手写，~~之后版本可能会加上~~。