构建环境

需要至少支持C++17标准的编译器编译。

我们使用的编译器为`gcc8.1`，编译参数如下

```
g++ -O3 -I . -m64 -Wno-literal-suffix -Werror=return-type
```

我们的构建方法中还有Makefile，因此还需要make

我们在win10上编写及构建运行

使用说明

我们将大整数运算封装得跟基本整型类型的操作几乎相同，没有特别的操作。

由于c++没有内置大数库，同时也没有我们想要的方便使用并且有一定性能的第三方库，因此尝试编写了这个项目

头文件及链接

在使用该大整数运算，需要先包含我们的头文件`#include<bgn.h>`。

在编译时需要同时加入`bgn.cpp`，例如在`main.cpp`中使用，则

```
g++ -O3 -I .  main.cpp bgn.cpp -o a.exe
```

其中`-O3`是可选的

创建一个大整数对象

```
bgn a=0; //与普通整型相同
bgn b=-1; //支持负数
bgn c=1234567890987LL; // long long 类型
bgn d=11111111111111111111111111111111111111bgn; //大整数字面量，以bgn为后缀，可以带负号
bgn e="22222222222222222222222222222222"; //c风格字符串初始化
bgn f=string("-33333333333333333333333333333333");//string初始化
```

使用方法

支持运算操作`+ - * / % += -= *= /= %=`

支持取负操作`-`

额外增加了求幂方法`pow`，但参数只接受`uint64_t`

```
auto x=(2bgn).pow(100);
```

支持比较运算`== != < <= > >=`

同时额外增加`cmp`方法，避免反复比较，`cmp`将会返回一个`int`值

```
f(const bgn &x,const bgn &y)
{
    int c=x.cmp(y);
    if(c<0)     //cmp返回值小于0则表示x<y
        ...
    else if(c>0)//cmp返回值大于0则表示x>y
        ...
    else        //cmp返回值等于0则表示x==y
        ...
}
```

增加了强制类型转换，可以将大整数转换成基本整型

`int32_t()`会取大整数的低31位

`int64_t()`会取大整数的低63位

```
bgn x=(9bgn).pow(9)
int a=int(x);
```

拥有`to_string`方法，产生一个十进制表示的`string`对象，主要用于输出

同时重载了`cout<<`，用于输出到终端

```
bgn x=1;
cout<<x<<std::endl;
```

使用注意事项

1. 除法的实现并不完善，目前只完成短除法，主要用于转换成10进制
2. 操作`+=`与`-=`会在原始对象上进行操作，因此尽量将`x=x+y`写成`x+=y`以减少临时对象的产生，对于乘除等其他运算无法避免临时对象生成，不需要注意这个问题
3. 在初始化`bgn`对象时，无论何种初始化方式，最终均由`bgn(string)`构造，这是为了简化负数转换的过程，因为`bgn`建立在无符号大数运算之上，基本单元（我们采用的是`uint32_t`）最小的负数（补码）的绝对值无法由基本单元表示，在强制类型转换时我们的实现同样不能取得`int64_t`的最小值。大部分时候这些开销可以忽略，因此不需过多关注。

使用样例

计算3的1000次方减2的1000次方，并输出

```
auto x=(3bgn).pow(1000)-(2bgn).pow(1000);
cout<<x<<endl;
```

计算100000的阶乘，并输出

```
bgn s=1;
for(int i=1;i<=100000;i++)
    s*=i;
cout<<s<<endl;
```