LAB1_REPORT

任务一：

考虑将乘法转换成加法: $a \times b = \sum_{i=1}^{b} a$, 加法每次超过m时，就mod m一下，则代码如下：

```
int64_t multimod_p1(int64_t a, int64_t b, int64_t m) {
  int64_t r=0;
  for(int i=0;i<b;i++){
    r+=a;
    if(r>m) r%=m;
  }
  return r;
}
```

这一算法需要循环b次，时间开销极大(在与下面改进代码和神秘代码相比要多100000倍)，所以测试一次(1000000次调用)所需时间贼长，请看下面的优化代码。

任务二：

由提示知：
$a \cdot b \% m = \sum_{i=0}^{62} b_i a \times 2^i \% m = a \times b_0 + 2 \times (a \times b_1 + (2 \times a \times b_2 + \ldots + (2 \times a \times b_{62}))) \% m$, 故考虑以下快速幂算法：

```
int64_t multimod_p2(int64_t a, int64_t b, int64_t m) {
  int64_t r=0;
  uint64_t byte=0x1;
  uint64_t temp[63];
  for(int i=0;i<63;i++){
    if(i==0){
      temp[i]=b%m;
      continue;
    }
    temp[i]=(((uint64_t)temp[i-1])<<1)%m;
  }
  for(int i=0;i<63;i++){
    if(byte&a) r=(r+temp[i])%m;
    byte<<=1;
  }
  return r;
}
```

在测试中，我先是用python写了一个生成随机数的程序并将结果写入txt文件:

```
import random
import os

with open("./data.txt","w") as f:
    for i in range(1000000):
        a=random.randint(1,2**63-1)
        b=random.randint(1,2**63-1)
        m=random.randint(1,2**63-1)
        res=a*b%m
        f.writelines(str(a)+' '+str(b)+' '+str(c)+' '+str(res)+'\n')
f.close()
```

然后在框架代码中添加了一个测试文件text1.c:

```
int64_t myatoi(char *p){
  int64_t res=0;
  while(*p!='\0'){
    res=res*10+(*p++-'0');
    if(*p=='\r') break;
  }
  return res;
}
int main(){
  FILE *fp=fopen("filename","r");
  char buf[100];
  int begin,end;
  double t1=0,t2=0,t3=0;
  while(fgets(buf,sizeof(buf),fp)){
    char *p=strtok(buf," ");
    int64_t a=myatoi(p);
    p=strtok(NULL," ");
    int64_t b=myatoi(p);
    p=strtok(NULL," ");
    int64_t m=myatoi(p);
    p=strtok(NULL,"\n");
    int64_t res=myatoi(p);
    //printf("%ld,%ld,%ld,%ld\n",a,b,m,res);
    //测试p1
    begin=clock();
    int64_t res1=multimod_p1(a,b,m);
    end=clock();
    t1+=end-begin;
    //测试p2
    begin=clock();
    int64_t res2=multimod_p2(a,b,m);
    end=clock();
    t2+=end-begin;
    //测试p3
    begin=clock();
    int64_t res3=multimod_p3(a,b,m);
    end=clock();
    t3+=end-begin;
    //int64_t res1=res2;
    if(res1!=res || res2 !=res || res3!=res){
      printf("res1 is %ld,res2 is %ld,res3 is %ld\n,res is
%ld.\n",res1,res2,res3,res);
```

```
      assert(0);
    }
  }
  fclose(fp);
  t1/=CLOCKS_PER_SEC;
  t2/=CLOCKS_PER_SEC;
  t3/=CLOCKS_PER_SEC;
  printf("time1 is %f seconds,time2 is %f seconds,time3 is %f
seconds\n",t1,t2,t3);
  return 0;
}
```

其中myatoi是用来将文件中的字符串转换成数字，因为C库中的atoi是int型的，将无法转换64位的整数。数据文件中共有1000000条数据，因为发现神秘代码在数较大时会失效，故同时测试三个代码时将python随机数生成限定在2^31-1范围内，由于上文提到的原因，不妨将multimod_p1的时间置为∞。计算函数运行时间使用了clock()函数。以下为10次生成随机数下此范围内的测试时间均值：

time/s	O0	O1	O2
multimod_p1	∞	∞	∞
multimod_p2	1.488664	1.486044	1.475539
multimod_p3	0.400260	0.399848	0.397747

在2^63-1范围内仅测试multimod_p2，发现它一直正确，10次生成随机数下的测试时间均值结果为：

time/s	O0	O1	O2
multimod_p2	1.698858	1.697477	1.695724

可以看出，O0，O1，O2优化稍微能减少一些时间消耗，但几乎可以不计。
性能分析：
p1需要循环b次，时间复杂度为O(b)，b为一个大整数，时间复杂度极高。
p2需循环63次，时间复制度为O(1),但其常数为63，时间复杂度较低。
p3仅需O(1)时间，是三个里最快的一个算法,比p2快了约3倍。

任务三：

神秘代码原理为：$a \times b\%m = (a \times b - (a \times b/m) \times m)\%m$,将a强制转出双精度double型，但除以m之后转出int64_t相当与抵消了精度损失，所以是有效的。
但在实验中发现神秘代码并非一直有效，当ab*过大时，它不能得出正确的答案。64位双精度表示为：0，1~11，12~63*
*但当*ab过大时(大于2^32),会导致double精度表示不足，造成误差。