1. 程式碼架構

因為上次幾乎已將 CRUD 模組化，所以只需把下面五個函式寫入.C 檔再進行呼叫就可以。

```
link deletenode(link head, link ptr);
link createlink(link head, char *key, char *value);
void updateNodeValue(link node, char *newValue);
link findnode(link head, char *key);
void freelist(link head);
```

因為這是只需要用到寫入和讀取，所以這次主程式只有用到 createlink 和 findode 兩個函式。

這次一共引入兩個檔案"myredis.c","hiredis/libhiredis.a"。前者為我寫得副程式，後面為 redis 的.a 檔。

```
//以自己設定的RANDOM方式，從0-9,A-Z,a-z選character
char randoml(int amount)
{
    amount=amount%62;
    if(amount<10) return ('0'+amount);
    else if(amount<36) return ('a' + amount - 10);
    else return ('A' + amount - 36);
}
```

圖一

Randoml 將輸入的數字%52，小於 10 為數字，大於 10 且小於 36 為小寫字母。大於 36 小於 52 為大寫字母。

```
void generate_data()
{
    for (int i=0;i<5;i++)
    {
        for (int j=0;j<62;j++)
        {
            for (int k=0;k<62;k++)
            {
                for (int l=0;l<62;l++)
                {
                    if(i*62*62*62+j*62*62+k*62+l<MILLION)
                    {
                        data[i*62*62*62+j*62*62+k*62+l][0]=randoml(i);
                        data[i*62*62*62+j*62*62+k*62+l][1]=randoml(j);
                        data[i*62*62*62+j*62*62+k*62+l][2]=randoml(k);
                        data[i*62*62*62+j*62*62+k*62+l][3]=randoml(l);
                        data[i*62*62*62+j*62*62+k*62+l][4]='\0';
                    }
                    else break;
                }
```

圖二

Gererate_data 以四層迴圈的數字 ijkl 當作輸入 randoml 裡面，組合後得到一個字串，以此來產生一百萬筆資料。

```c
void benchmark_get(link head, redisContext *c, double *time, int *spaces) {
    int i;
    link ptr1 = head;
    char key[8];
    clock_t start, finish;
    //以下為測試讀取時間
    start = clock();
    for (i = 0; i < MILLION; i++) {
        sprintf(key, "%d", i);
        redisReply *reply = redisCommand(c, "GET %s", key);
        if (reply) {
            freeReplyObject(reply);
        }
    }
    finish = clock();
    time[3] = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("Total time for getting %d data need %.6fs\n", i, time[3]);
    printf("Average time for getting %d one data need %.6fus\n", i, (double)time[3] / i * MILLION);
}
```

圖三

Benchmark_get 是把 redis 的 read 寫成 function，start 為開始的時間，然後將 key 慢慢傳進去做讀取，最後 finish 為結束時間，兩者相減為 redis read 的時間。

`void benchmark_insert(link head,redisContext *c,double *time,int *spaces)`

包含 benchmark_get，先進行 myredis 寫入的時間和記憶體測量大小，分別存入 time[0]和 spaces[0]，再進行讀取的時間計算並 print 出以上數據。

之後進行 redis 寫入的時間和記憶體測量大小，再通過呼叫 benchmark_get 得到 redis 讀取時間的數據，之後將上述都 print 出來。

```c
    //連線REDIS
    redisContext *c = redisConnect("127.0.0.1", 6379);
    if (c == NULL || c->err)
    {
        if (c)
        {
            printf("Error: %s\n", c->errstr);
        // handle error
        }
        else printf("Can't allocate redis context\n");
    }
```

圖四

主程式主要進行 redis 連線，之後呼叫 benchmark_insert 完成所有操作。我還在跑完數據後可以進行 myredis 的讀取操作。

程式輸出結果說明

```
Benchmarks for myredis:
Total time for inserting 1000000 data need 0.129010s
Avergae time for inserting 1000000 data need 0.129010us
Total memory for inserting 1000000 data need 96000000bytes
Avergae memory for inserting 1000000 data need 96bytes
Total time for getting 1000000 data need 4528.464014s
Avergae time for getting 1000000 data need 4.528464ms

Benchmarks for hiredis:
Total time for inserting 1000000 data need 40.902771s
Avergae time for inserting 1000000 one data need 40.902771us
Total time for getting 1000000 data need 40.316374s
Average time for getting 1000000 one data need 40.316374us
Total memory for inserting 1000000 datas need 65231248bytes
Avergae memory for inserting 1000000 one data need 65bytes
```

圖五

Myredis

平均寫入:0.129us

平均讀取:4.526ms

記憶體總量:96000000 bytes

Redis:

平均寫入:40.9us

平均讀取:40.3us

記憶體總量:65231248 bytes