Project 2
机场调度

班级：教务 2 班

宋渝杰 18340146

廖家源 18340105

缪贝琪 18340131

刘依澜 18340121

【题目要求】

P1：整合 pdf 给出的用于飞机场模拟的函数和方法，形成一个完整的程序，并用此模拟程序做多次实验，调整着陆和起飞的飞机数的期望值，找出在飞机不会被拒绝服务的条件下这些数字的尽可能的最大值。如果队列的长度增加或减少，分析这些值的变化。

P2：修改 P1，使飞机场有两条跑道，一条用于着陆，一条用于起飞。与 P1 比较相应数字，分析是否为 P1 的两倍。

P3：修改 P1，使飞机场有两条跑道，一条用于着陆，一条用于起飞。如果某个队列是空的，则两条跑道均可用于着陆或起飞，着陆优先起飞。

P4：修改 P1，使飞机场有三条飞机跑道，一条用于着陆，一条用于起飞。第三条用于着陆，当着陆队列为空时，第三条也可以用于起飞。

P5：修改 P1，使得每架飞机到达时有一个随机产生的油量，如果油量不足则允许它优先着陆。

P6：修改 P1，写一个占位程序代替随机数函数，使得用户可以精准的控制每一个时间单元内每个队列到达的飞机数。

【数据结构与算法】

数据结构：

（1） 类 Runway，用于抽象出机场 small_airport，具有着陆轨道、起飞轨道、各种统计的属性，具有相应的构造函数、判断着陆和起飞的函数、统计数据的函数。

（2） 类 Plane，用于抽象出飞机 plane，具有飞机编号、到达时间（P5 还具有油量）的属性，具有相应的构造函数、着陆、起飞、前往另一个机场的函数（P5 还具有油量不足而坠毁的函数）。

（3） 类 Queue、Extended_queue，用于抽象出着陆和起飞的队列，具有队列长度、头尾位置的属性，具有相应的构造函数、服务飞机函数、判断是否为满、空队列，返回队列长度的函数。

（4） 类 Random，用于提供满足泊松分布的函数，返回每个时间着陆和起飞的飞机数量，以及提供 P6 需要的占位函数。

算法：

（1） 程序流程：进入程序中显示菜单，菜单中选择 P1-P6 模式，P1-P5 中根据程序提示输入队列长度、模拟时长、平均着陆和起飞的飞机的数量（P6 根据系统提示输入队列长度、模拟时长，之后每一个时间段由用户输入着陆和起飞飞机的数量），程序经过模拟之后，显示每一段时间的飞机着陆和起飞的事件，最后显示统计数据。之后回到主菜单。

（2） 大部分的算法（飞机降落、起飞的判断和实现）来自于 pdf 内提供的函数，其他需

要自行实现的算法有：

P1-P6：泊松分布函数：输入平均值返回一个整数，随着调用次数增加，返回整数的平均值趋近于给定的平均值。

P2：small_airport.activity2()：实现一条跑道用于着陆，另一条跑道用于起飞的操作。

P3：small_airport.activity3()：在 P2 基础上实现当着陆或起飞队列为空时，两条跑道均可用于着陆或起飞，着陆优先起飞。

P4：small_airport.activity4()：实现三条跑道的着陆起飞操作。

P5：small_airport.activity5()：实现油量低的优先着陆操作。

P6：Random::set(int current_time)：实现指定时刻用户自行输入着陆和起飞的飞机数量的操作。

（3） 实现方式：

1. 泊松分布：此处参考高德纳的算法（如图 1），转化为 c++代码得到：

```
algorithm poisson random number (Knuth):
    init:
        Let L ← e^-λ, k ← 0 and p ← 1.
    do:
        k ← k + 1.
        Generate uniform random number u in [0,1] and let p ← p×u.
    while p > L.
    return k − 1.
```

图 1 高德纳泊松分布算法

测试发现，在输入期望值较小时（3 以内），近万次返回值平均后平均值与期望值十分接近。

2. activity2-5 函数：分别判断着陆队列和起飞队列的长度：

P2：只要着陆队列不空，即允许一台飞机着陆，只要起飞队列不空，即允许一台飞机起飞，当全空时，提示跑道闲置。

P3：先判断着陆队列和起飞队列的情况，然后以 着陆+起飞 > 双着陆 > 双起飞 > 单着陆或起飞 的优先度对飞机进行操作。

P4：优先度为双着陆+起飞 > 着陆+双起飞 > 着陆+起飞 > 双着陆 > 双起飞 > 单着陆或起飞 的优先度对飞机进行操作。

P5：以 P1 为基础，着陆优先起飞，油量低的优先着陆。

3. Random::set(int current_time)：通过传入时间，函数提示用户输入这个时间着陆和起飞的飞机数，以 pair<int,int>形式将输入的着陆和起飞的飞机数返回。

菜单显示：

打开程序后，显示操作菜单，程序提示用户输入相应编号，以进入 P1-P6 其中一个模式，输入 q 则程序结束。

异常处理：

当用户输入错误信息时（例如该输入数字时，用户输入了字符），系统要么提示输入错误，请重新输入，然后让用户重新输入；要么不予接受用户的输入，使用清理函数清理用户的错误输入，并让用户重新输入。

【测试数据、结果及分析】

菜单页面：（如图 2）

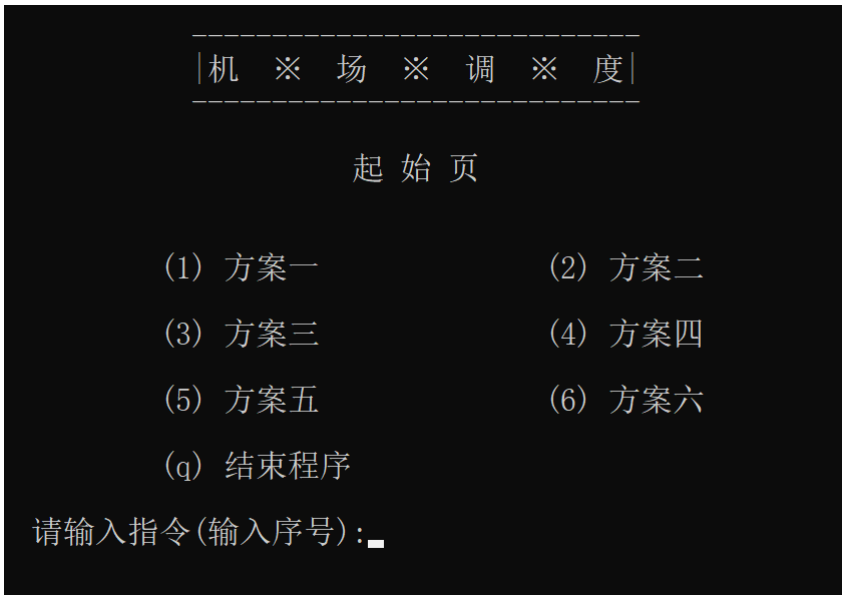

图 2 菜单页面

P1：在菜单中输入数字 1 即进入 P1 页面：

输入 pdf 中给的输入样例（如图 3-1），模拟结束之后得到统计数据（如图 3-2）：

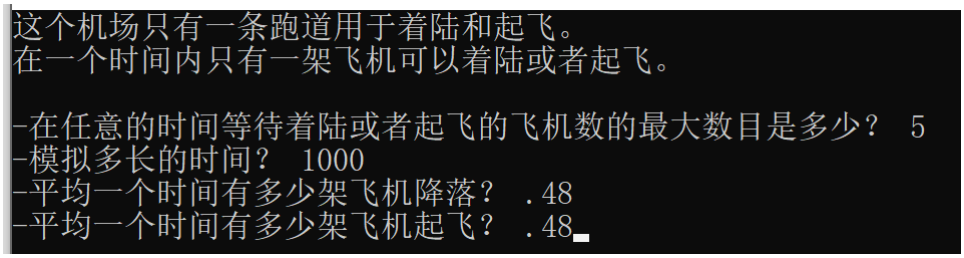

图 3-1 P1 输入过程

```
模拟在 1000 时间后结束。
  处理的飞机数量：  955
  要求着陆的飞机数量：  486
  要求起飞的飞机数量：  469
  接受着陆的飞机数量：  484
  接受起飞的飞机数量：  420
  拒绝着陆的飞机数量：  2
  拒绝起飞的飞机数量：  49
  着陆的飞机数量：  484
  起飞的飞机数量：  418
  留在着陆队列里的飞机数量：  0
  留在起飞队列里的飞机数量：  2
  飞机跑道空闲的时间百分比：  9.8%
  飞机着陆的平均等待时长：  0.423554
  飞机起飞的平均等待时长：  4.34689
  希望着陆的飞机平均被观察率为：  每单位时间 0.486 架。
  希望起飞的飞机平均被观察率为：  每单位时间 0.469 架。
```

图 3-2 P1 统计数据

分析：

1. 统计结果与 pdf 给出的统计结果相近。

2. 当队列长度为 5 时，平均着陆率和起飞率为 0.32 时，飞机基本不会被拒绝服务。大于 0.32 则出现拒绝服务几率大大增加。

3. 队列长度增加，平均着陆率和起飞率也会增加。例如当队列长度增加至 20 时，平均着陆率和起飞率可增加至 0.4，飞机基本不会被拒绝服务。

P2：在菜单中输入数字 2 即进入 P2 页面：

设置队列长度为 5，多次测试（实验结果如图 4）。

```
模拟在 1000 时间后结束。
  处理的飞机数量：  1195
  要求着陆的飞机数量：  587
  要求起飞的飞机数量：  608
  接受着陆的飞机数量：  587
  接受起飞的飞机数量：  607
  拒绝着陆的飞机数量：  0
  拒绝起飞的飞机数量：  1
  着陆的飞机数量：  587
  起飞的飞机数量：  607
  留在着陆队列里的飞机数量：  0
  留在起飞队列里的飞机数量：  0
  飞机跑道空闲的时间百分比：  16%
  飞机着陆的平均等待时长：  0.58092
  飞机起飞的平均等待时长：  0.673806
  希望着陆的飞机平均被观察率为：  每单位时间 0.587 架。
  希望起飞的飞机平均被观察率为：  每单位时间 0.608 架。
```

图 4 P2 测试结果

得到结果：当平均着陆率和起飞率为 0.64 时，飞机基本不会被拒绝服务，继续增加至 0.7 时，有几架飞机被拒绝服务，因此得到结论：P2 相应数字确实是 P1 的两倍左右。

P3-P6 只要求实现程序，不要求对数据情况进行分析，在此只附上 P3-P6 程序测试图：

```
这个机场有两条跑道用于着陆和起飞。
在一个时间内有一架飞机可以着陆，有一架飞机可以起飞。
如果某个队列是空的，那么两条跑道均可用于着陆或均可用于起飞。

-在任意的时间等待着陆或者起飞的飞机数的最大数目是多少？  5
-模拟多长的时间？  1000
-平均一个时间有多少架飞机降落？  .48
-平均一个时间有多少架飞机起飞？  .48
```

图 5-1 P3 输入过程

```
模拟在 1000 时间后结束。
  处理的飞机数量：  923
  要求着陆的飞机数量：  464
  要求起飞的飞机数量：  459
  接受着陆的飞机数量：  464
  接受起飞的飞机数量：  459
  拒绝着陆的飞机数量：  0
  拒绝起飞的飞机数量：  0
  着陆的飞机数量：  464
  起飞的飞机数量：  459
  留在着陆队列里的飞机数量：  0
  留在起飞队列里的飞机数量：  0
  飞机跑道空闲的时间百分比：  36.7%
  飞机着陆的平均等待时长：  0.142241
  飞机起飞的平均等待时长：  0.117647
  希望着陆的飞机平均被观察率为：  每单位时间 0.464 架。
  希望起飞的飞机平均被观察率为：  每单位时间 0.459 架。
```

图 5-2 P3 测试结果

```
这个机场有三条跑道用于着陆和起飞。
在一个时间内有一条跑道可以着陆，有一条跑道可以起飞。
第三条跑道用于着陆，如果没有飞机着陆，也可以用于起飞。

-在任意的时间等待着陆或者起飞的飞机数的最大数目是多少？  5
-模拟多长的时间？  1000
-平均一个时间有多少架飞机降落？  .48
-平均一个时间有多少架飞机起飞？  .48
```

图 5-3 P4 输入过程

```
模拟在 1000 时间后结束。
  处理的飞机数量:  943
  要求着陆的飞机数量:  471
  要求起飞的飞机数量:  472
  接受着陆的飞机数量:  471
  接受起飞的飞机数量:  472
  拒绝着陆的飞机数量:  0
  拒绝起飞的飞机数量:  0
  着陆的飞机数量:  471
  起飞的飞机数量:  472
  留在着陆队列里的飞机数量:  0
  留在起飞队列里的飞机数量:  0
  飞机跑道空闲的时间百分比:  37%
  飞机着陆的平均等待时长:  0.0233546
  飞机起飞的平均等待时长:  0.029661
  希望着陆的飞机平均被观察率为:  每单位时间 0.471 架。
  希望起飞的飞机平均被观察率为:  每单位时间 0.472 架。
```

图 5-4 P4 测试结果

```
这个机场只有一条跑道用于着陆和起飞。
在一个时间内只有一架飞机可以着陆或者起飞。
考虑飞机燃料剩余量，优先让剩余量少的降落。

-在任意的时间等待起飞的飞机数的最大数目是多少?  5
-模拟多长的时间?  1000
-平均一个时间有多少架飞机降落?  .48
-平均一个时间有多少架飞机起飞?  .48
```

图 5-5 P5 输入数据

```
模拟在 1000 时间后结束。
  处理的飞机数量:  946
  要求着陆的飞机数量:  466
  要求起飞的飞机数量:  480
  接受着陆的飞机数量:  466
  接受起飞的飞机数量:  426
  坠毁的飞机数量:  0
  拒绝起飞的飞机数量:  54
  着陆的飞机数量:  466
  起飞的飞机数量:  424
  留在着陆队列里的飞机数量:  0
  留在起飞队列里的飞机数量:  2
  飞机跑道空闲的时间百分比:  11%
  飞机着陆的平均等待时长:  0.328326
  飞机起飞的平均等待时长:  4.61792
  希望着陆的飞机平均被观察率为:  每单位时间 0.466 架。
  希望起飞的飞机平均被观察率为:  每单位时间 0.48 架。
```

图 5-6 P5 测试结果

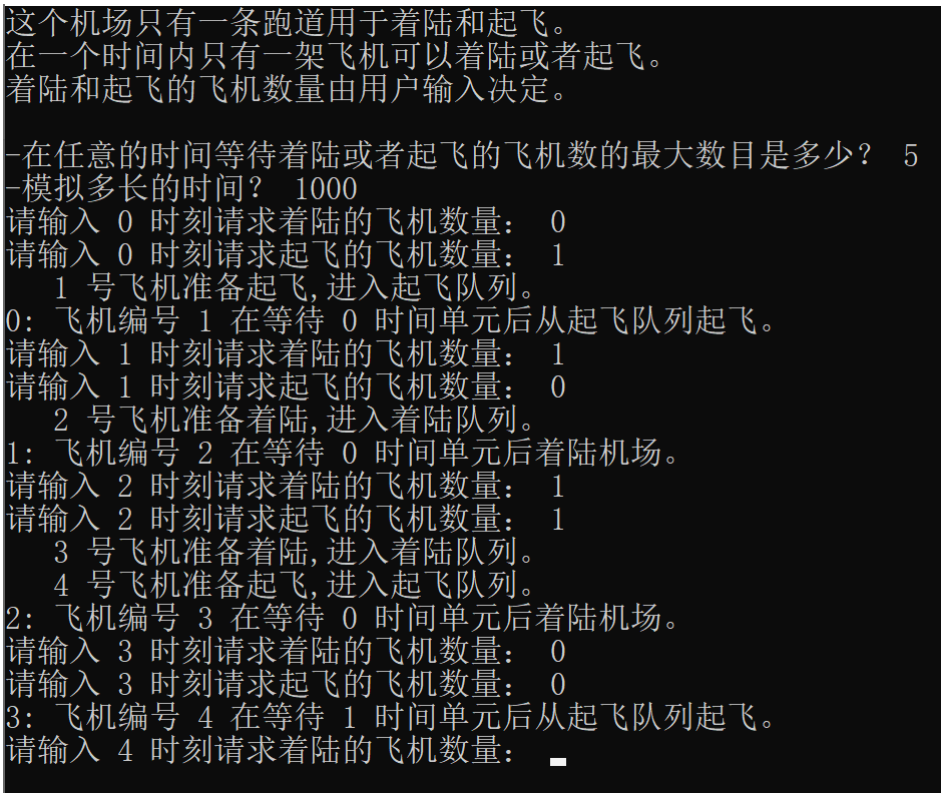

图 5-7 P6 测试结果

注：P6 是在 P1 基础上手动输入着陆和起飞的飞机数量，在此只做少量输入的显示。

【分工、贡献%、自我评分】

分工：

宋渝杰：P1、P3、P6 的实现，数据测试和分析 30% 10 分

廖家源：P5、菜单页面、异常处理的实现，界面美化工作 30% 10 分

缪贝琪：P4 的实现、编写实验报告 20% 9 分

刘依澜：P2 的实现、编写实验报告 20% 9 分

【项目总结】

从这次的小组实验中，由于工作量和难度比项目一要大上许多，一两个人完成主体部分变得更加艰难，所以更加强调团队同心协力合作，共同解决问题。

在这个合作的过程中也不免会出现一些问题，例如项目分几个部分，队友完成相应部分后对整个项目的整合，以及增强自己的代码可读性，读懂队友编写的代码等等。

总的项目完成之后，对努力之后的成果还算比较满意，从中我们也加强了对多文件的规范写法、类的实例化以及应用，以及模拟之后得到的数据进行分析的能力，等等。

在下一次项目之前，我们会更加强调团队意识，例如积极带动能力相对薄弱的同学抓紧时间提高编程技术，给她们分配力所能及的任务，以及提高她们的时间意识和任务意识等等。

【程序清单】

这次项目采用多文件写法，共13个文件：

Function.hpp: //共18行代码

```
#ifndef Function_hpp
#define Function_hpp
#include<iostream>
using namespace std;

void initialize(int &end_time, int &queue_limit,double &arrival_rate, double &departure_rate,int& max);
void initialize(int &end_time, int &queue_limit);
void run_idle(int time);
void Base_Print();
void Menu();
void Subpro1();
void Subpro2();
void Subpro3();
void Subpro4();
void Subpro5();
void Subpro6();

#endif
```

Function.cpp: //共447行代码

```
#include<iostream>
#include<cmath>
#include<string>
#include<conio.h>
#include<windows.h>

#include"Function.hpp"
#include"Plane.hpp"
#include"Runway.hpp"
#include"Random.hpp"
using namespace std;
```

```
void initialize(int &end_time, int &queue_limit,double &arrival_rate, double &departure_rate,int flag){ //P1-P5 的输入函数
    if(flag) cout << "-在任意的时间等待着陆或者起飞的飞机数的最大数目是多少？ "<< flush;
    else cout << "-在任意的时间等待起飞的飞机数的最大数目是多少？ "<< flush;
     CI1:
  fflush(stdin);
    cin >> queue_limit;
    if(!cin.good() or queue_limit<=0){
     cin.clear(); cin.sync();
     cerr << "--输入错误,请重新输入: ";
     goto CI1;
    }

    cout << "-模拟多长的时间？ " << flush;
    CI2:
    fflush(stdin);
  cin >> end_time;
    if(!cin.good() or end_time<=0){
     cin.clear(); cin.sync();
     cerr << "--输入错误,请重新输入: ";
     goto CI2;
    }

    cout << "-平均一个时间有多少架飞机降落？ " << flush;
    CI3:
    fflush(stdin);
  cin >> arrival_rate;
    if(!cin.good() or arrival_rate<=0){
     cin.clear(); cin.sync();
     cerr << "--输入错误,请重新输入: ";
     goto CI3;
    }
    cout << "-平均一个时间有多少架飞机起飞？ " << flush;
```

```
    CI4:
    fflush(stdin);
  cin >> departure_rate;
    if(!cin.good() or departure_rate<=0){
     cin.clear(); cin.sync();
     cerr << "--输入错误,请重新输入: ";
     goto CI4;
    }


}

void initialize(int &end_time, int &queue_limit){ //P6 输入函数

    cout << "-在任意的时间等待着陆或者起飞的飞机数的最大数目是多少？ "<< flush;
    CL1:
    fflush(stdin);
  cin >> queue_limit;
  if(!cin.good() or queue_limit<=0){
     cin.clear(); cin.sync();
     cerr << "--输入错误,请重新输入: ";
     goto CL1;
    }
    cout << "-模拟多长的时间？ " << flush;
    CL2:
  fflush(stdin);
    cin >> end_time;
    if(!cin.good() or end_time<=0){
     cin.clear(); cin.sync();
     cerr << "--输入错误,请重新输入: ";
     goto CL2;
    }
}

void run_idle(int time){
```

```
    cout << time << ": 跑道是闲置的。 " << endl;
}

void Menu(){ //菜单显示
  Base_Print();
  cout<<"\n\t\t\t\t\t\t\t    起 始 页";
  cout<<"\n\n\n";
  cout<<"\t\t\t\t\t(1) 方案一";
  cout<<"\t\t(2) 方案二\n\n";
  cout<<"\t\t\t\t\t(3) 方案三";
  cout<<"\t\t(4) 方案四\n\n";
  cout<<"\t\t\t\t\t(5) 方案五";
  cout<<"\t\t(6) 方案六\n\n";
  cout<<"\t\t\t\t\t(q) 结束程序\n";
  cout<<"\n\t\t\t\t请输入指令(输入序号):";
}

void Base_Print(){
  cout<<"\n";
  cout<<"\t\t\t\t\t  ----------------------------"<<endl;
  cout<<"\t\t\t\t\t  |机  ※  场  ※  调  ※  度|"<<endl;
  cout<<"\t\t\t\t\t  ----------------------------"<<endl;
}

void Subpro1(){ //P1
  int end_time;
    int queue_limit;
    int flight_number = 1;
    double arrival_rate, departure_rate;
    Random variable;

  cout << "这个机场只有一条跑道用于着陆和起飞。" << endl
     << "在一个时间内只有一架飞机可以着陆或者起飞。" << endl<< endl;;
    srand((unsigned)time(NULL));
  initialize(end_time, queue_limit, arrival_rate, departure_rate,true);
```

```
    Runway small_airport(queue_limit);

  for (int current_time = 0; current_time < end_time; current_time++) {
//         飞机降落
        int number_arrivals = variable.poisson(arrival_rate);
        for (int i = 0; i < number_arrivals; i++) {
            Plane current_plane(flight_number++, current_time, arriving);
            if (small_airport.can_land(current_plane) != success)
                current_plane.refuse();
            else
            current_plane.wait();
        }
//         飞机起飞
        int number_departures = variable.poisson(departure_rate);
        for (int j = 0; j < number_departures; j++) {
         Plane current_plane(flight_number++, current_time, departing);
            if (small_airport.can_depart(current_plane) != success)
                current_plane.refuse();
            else
            current_plane.wait();
        }
        Plane moving_plane;
//         跑道选择着陆或起飞
        switch (small_airport.activity1(current_time, moving_plane)) {
            case land:
                        moving_plane.land(current_time);
                        break;
            case takeof:
                        moving_plane.fly(current_time);
                        break;
            case idle:
                        run_idle(current_time);
        }
    }
//    结束模拟并总结
```

```
    small_airport.shut_down(end_time);
  cout<<"\n\t\t\t\t\t\t\t\t按任意键返回";
  _getch();
}

void Subpro2(){ //P2
  int end_time;
    int queue_limit;
    int flight_number = 1;
    double arrival_rate, departure_rate;
    Random variable;

  cout << "这个机场有两条跑道用于着陆和起飞。" << endl
      << "在一个时间内有一架飞机可以着陆，有一架飞机可以起飞。" << endl<< endl;
    srand((unsigned)time(NULL));
  initialize(end_time, queue_limit, arrival_rate, departure_rate,true);
    Runway small_airport(queue_limit);

  for (int current_time = 0; current_time < end_time; current_time++) {
//        飞机降落
        int number_arrivals = variable.poisson(arrival_rate);
        for (int i = 0; i < number_arrivals; i++) {
            Plane current_plane(flight_number++, current_time, arriving);
            if (small_airport.can_land(current_plane) != success)
                current_plane.refuse();
            else
            current_plane.wait();
        }
//        飞机起飞
        int number_departures = variable.poisson(departure_rate);
        for (int j = 0; j < number_departures; j++) {
            Plane current_plane(flight_number++, current_time, departing);
            if (small_airport.can_depart(current_plane) != success)
                current_plane.refuse();
            else
```

```
            current_plane.wait();
        }
        Plane moving_plane,moving_plane2;
//          跑道选择着陆或起飞
        switch       (small_airport.activity2(current_time,       moving_plane,
moving_plane2)) {
            case land_and_takeof:
                moving_plane.land(current_time);
                moving_plane2.fly(current_time);
                break;
            case land:
                moving_plane.land(current_time);
                break;
            case takeof:
                moving_plane2.fly(current_time);
                break;
            case idle:
                run_idle(current_time);
        }
    }
//    结束模拟并总结
    small_airport.shut_down(end_time);
  cout<<"\n\t\t\t\t\t\t\t\t 按任意键返回";
  _getch();
}

void Subpro3(){ //P3
  int end_time;
    int queue_limit;
    int flight_number = 1;
    double arrival_rate, departure_rate;
    Random variable;

  cout << "这个机场有两条跑道用于着陆和起飞。 " << endl
       << "在一个时间内有一架飞机可以着陆，有一架飞机可以起飞。 " << endl
```

```
        << "如果某个队列是空的，那么两条跑道均可用于着陆或均可用于起飞。 " <<
endl<< endl;
    srand((unsigned)time(NULL));
  initialize(end_time, queue_limit, arrival_rate, departure_rate,true);
    Runway small_airport(queue_limit);

  for (int current_time = 0; current_time < end_time; current_time++) {
//        飞机降落
        int number_arrivals = variable.poisson(arrival_rate);
        for (int i = 0; i < number_arrivals; i++) {
            Plane current_plane(flight_number++, current_time, arriving);
            if (small_airport.can_land(current_plane) != success)
                current_plane.refuse();
             else
            current_plane.wait();
        }
//        飞机起飞
        int number_departures = variable.poisson(departure_rate);
        for (int j = 0; j < number_departures; j++) {
            Plane current_plane(flight_number++, current_time, departing);
            if (small_airport.can_depart(current_plane) != success)
                current_plane.refuse();
             else
            current_plane.wait();
        }
        Plane moving_plane,moving_plane2;
//        跑道选择着陆或起飞
        switch       (small_airport.activity3(current_time,       moving_plane,
moving_plane2)) {
            case land_and_takeof:
                moving_plane.land(current_time);
                moving_plane2.fly(current_time);
                break;
            case land:
                moving_plane.land(current_time);
```

```
                break;
            case takeof:
                moving_plane2.fly(current_time);
                break;
            case land2:
                moving_plane.land(current_time);
                moving_plane2.land(current_time);
                break;
            case takeof2:
                moving_plane2.fly(current_time);
                moving_plane.fly(current_time);
                break;
            case idle:
                run_idle(current_time);
        }
    }
//    结束模拟并总结
    small_airport.shut_down(end_time);
  cout<<"\n\t\t\t\t\t\t\t\t 按任意键返回";
  _getch();
}

void Subpro4(){ //P4
  int end_time;
    int queue_limit;
    int flight_number = 1;
    double arrival_rate, departure_rate;
    Random variable;

  cout << "这个机场有三条跑道用于着陆和起飞。 " << endl
        << "在一个时间内有一条跑道可以着陆，有一条跑道可以起飞。 " << endl
        << "第三条跑道用于着陆，如果没有飞机着陆，也可以用于起飞。 " << endl<<
endl;
    srand((unsigned)time(NULL));
  initialize(end_time, queue_limit, arrival_rate, departure_rate,true);
```

```
    Runway small_airport(queue_limit);
  for (int current_time = 0; current_time < end_time; current_time++) {
//         飞机降落
        int number_arrivals = variable.poisson(arrival_rate);
        for (int i = 0; i < number_arrivals; i++) {
            Plane current_plane(flight_number++, current_time, arriving);
            if (small_airport.can_land(current_plane) != success)
                current_plane.refuse();
             else
            current_plane.wait();
        }
//         飞机起飞
        int number_departures = variable.poisson(departure_rate);
        for (int j = 0; j < number_departures; j++) {
            Plane current_plane(flight_number++, current_time, departing);
            if (small_airport.can_depart(current_plane) != success)
                current_plane.refuse();
             else
            current_plane.wait();
        }
        Plane moving_plane,moving_plane2,moving_plane3;
//         跑道选择着陆或起飞
        switch       (small_airport.activity4(current_time,       moving_plane,
moving_plane2 ,moving_plane3)) {
            case land2_and_takeof:
                moving_plane.land(current_time);
                moving_plane2.land(current_time);
                moving_plane3.fly(current_time);
                break;
            case land_and_takeof2:
                moving_plane.land(current_time);
                moving_plane2.fly(current_time);
                moving_plane3.fly(current_time);
                break;
            case land_and_takeof:
```

```
                moving_plane.land(current_time);
                moving_plane2.fly(current_time);
                break;
            case land:
                moving_plane.land(current_time);
                break;
            case takeof:
                moving_plane2.fly(current_time);
                break;
            case land2:
                moving_plane.land(current_time);
                moving_plane2.land(current_time);
                break;
            case takeof2:
                moving_plane2.fly(current_time);
                moving_plane.fly(current_time);
                break;
            case idle:
                run_idle(current_time);
        }
    }
//    结束模拟并总结
    small_airport.shut_down(end_time);

  cout<<"\n\t\t\t\t\t\t\t\t 按任意键返回";
  _getch();
}

void Subpro5(){ //P5
  int end_time;
    int queue_limit;
    int flight_number = 1;
    double arrival_rate, departure_rate;
    Random variable;
```

```
  cout << "这个机场只有一条跑道用于着陆和起飞。" << endl
     << "在一个时间内只有一架飞机可以着陆或者起飞。" << endl
     <<"考虑飞机燃料剩余量，优先让剩余量少的降落。"<< endl<<endl;
    srand((unsigned)time(NULL));
  initialize(end_time, queue_limit, arrival_rate, departure_rate,false);
    Runway small_airport(queue_limit,1000);

  for (int current_time = 0; current_time < end_time; current_time++) {
//         飞机降落
        int number_arrivals = variable.poisson(arrival_rate);
        for (int i = 0; i < number_arrivals; i++) {
            Plane current_plane(flight_number++, current_time, arriving);
            small_airport.can_land2(current_plane);
            current_plane.wait2();
        }
//         飞机起飞
        int number_departures = variable.poisson(departure_rate);
        for (int j = 0; j < number_departures; j++) {
         Plane current_plane(flight_number++, current_time, departing);
            if (small_airport.can_depart(current_plane) != success)
                current_plane.refuse();
            else
            current_plane.wait();
        }
        Plane moving_plane;
//         跑道选择着陆或起飞
        switch (small_airport.activity5(current_time, moving_plane)) {
            case land:
                        moving_plane.land(current_time);
                        break;
            case takeof:
                        moving_plane.fly(current_time);
                        break;
            case idle:
                        run_idle(current_time);
```

```
        }
    }
//    结束模拟并总结
    small_airport.shut_down(end_time);
  cout<<"\n\t\t\t\t\t\t\t\t 按任意键返回";
  _getch();
}

void Subpro6(){ //P6
  int end_time;
    int queue_limit;
    int flight_number = 1;
    Random variable;

  cout << "这个机场只有一条跑道用于着陆和起飞。" << endl
     << "在一个时间内只有一架飞机可以着陆或者起飞。" << endl
     << "着陆和起飞的飞机数量由用户输入决定。" << endl << endl;
    srand((unsigned)time(NULL));
  initialize(end_time, queue_limit);
    Runway small_airport(queue_limit);
  for (int current_time = 0; current_time < end_time; current_time++) {
//        飞机降落
        pair<int, int> a = variable.set(current_time);
        int number_arrivals = a.first;
        for (int i = 0; i < number_arrivals; i++) {
            Plane current_plane(flight_number++, current_time, arriving);
            if (small_airport.can_land(current_plane) != success)
                current_plane.refuse();
             else
            current_plane.wait();
        }
//        飞机起飞
        int number_departures = a.second;
        for (int j = 0; j < number_departures; j++) {
            Plane current_plane(flight_number++, current_time, departing);
```

```
            if (small_airport.can_depart(current_plane) != success)
                current_plane.refuse();
             else
            current_plane.wait();
        }
        Plane moving_plane;
//         跑道选择着陆或起飞
        switch (small_airport.activity1(current_time, moving_plane)) {
            case land:
                        moving_plane.land(current_time);
                        break;
            case takeof:
                        moving_plane.fly(current_time);
                        break;
            case idle:
                        run_idle(current_time);
        }
    }
//    结束模拟并总结
    small_airport.shut_down(end_time);
  cout<<"\n\t\t\t\t\t\t\t\t 按任意键返回";
  _getch();
}
```

Extended_queue.hpp //共 19 行代码

```
#ifndef Extended_queue_hpp
#define Extended_queue_hpp

#include"Queue.hpp"

class Extended_queue: public Queue {
  public:
      Extended_queue();
      Extended_queue(int max);
      ~Extended_queue();
```

```
        bool full() const;
        int size() const;
        void clear();
        void insert(const Queue_entry &item);
        int go(int time);
        Error_code serve_and_retrieve(Queue_entry &item);
};

#endif
```

Extended_queue.cpp //共 66 行代码

```
#include"Extended_queue.hpp"
#include<iostream>
using namespace std;

Extended_queue::Extended_queue():Queue(){}

Extended_queue::Extended_queue(int max):Queue(max){}

Extended_queue::~Extended_queue(){}

bool Extended_queue::full() const{
    return count == maxqueue;
}

int Extended_queue::size() const{
    return count;
}

void Extended_queue::clear(){
    count = 0;
    rear = maxqueue - 1;
    front = 0;
}
```

```
void Extended_queue::insert(const Queue_entry &item){
     if(count==0 or item.getgas()>=entry[rear].getgas()){
         append(item);
     }else{
         for(int i=0;i<count;i++){
             if(item.getgas()<entry[(i+front)%maxqueue].getgas()){
                 for(int j=rear+1;j!=(i+front)%maxqueue;j--){
                     entry[j%maxqueue]=entry[(j+maxqueue-1)%maxqueue];
                 }
                 count++;
                rear = ((rear + 1) == maxqueue) ? 0 : (rear + 1);
                 entry[(i+front)%maxqueue]=item;
                 break;
             }
         }
    }
}

int Extended_queue::go(int time){
  Queue_entry item;
    int num=0;
    while(!empty()){
         if(entry[front].getgas()<=0){
             entry[front].fall(time);
             serve();
             num++;
         }else break;
     }
     for(int i=0;i<count;i++) entry[(i+front)%maxqueue].decgas();
     return num;
  }

Error_code Extended_queue::serve_and_retrieve(Queue_entry &item){
   if (count <= 0) return underflow;
    else{
```

```
        item = entry[front];
        count--;
        front = ((front + 1) == maxqueue) ? 0 : (front + 1);
        return success;
    }
}
```

Queue.hpp //共 24 行代码

```
#ifndef Queue_hpp
#define Queue_hpp

#include"Plane.hpp"
typedef Plane Queue_entry;

enum Error_code{ success, fail, underflow, overflow};
class Queue {
    public:
        Queue();
        Queue(int max);
        ~Queue();
        bool empty() const;
        Error_code serve();
        Error_code append(const Queue_entry &item);
        Error_code retrieve(Queue_entry &item) const;
    protected:
        int count;
        int front, rear;
        int maxqueue;
        Queue_entry* entry;
};

#endif
```

Queue.cpp //共 51 行代码

```
#include"Queue.hpp"
#include<iostream>
using namespace std;

Queue::Queue(){
  maxqueue = 0;
    count = 0;
    rear = maxqueue - 1;
    front = 0;
    entry = NULL;
}

Queue::Queue(int max){
  maxqueue = max;
    count = 0;
    front = 0;
    rear = maxqueue-1;
    entry = new Queue_entry[maxqueue];
}

Queue::~Queue(){
  if(entry!=NULL) delete []entry;
}

bool Queue::empty() const{
    return count == 0;
}

//尾部增加元素
Error_code Queue::append(const Queue_entry &item){
    if (count >= maxqueue) return overflow;
    count++;
    rear = ((rear + 1) == maxqueue) ? 0 : (rear + 1);
    entry[rear] = item;
    return success;
```

```
}

// 删除头部元素
Error_code Queue::serve(){
    if (count <= 0) return underflow;
    count--;
    front = ((front + 1) == maxqueue) ? 0 : (front + 1);
    return success;
}

//获取头部元素
Error_code Queue::retrieve(Queue_entry &item) const{
    if (count <= 0) return underflow;
    item = entry[front];
    return success;
}
```

Runway.hpp //共 39 行代码

```
#ifndef Runway_hpp
#define Runway_hpp

#include"Extended_queue.hpp"

enum Runway_activity {idle, land, land2, takeof, takeof2, land_and_takeof,
land2_and_takeof, land_and_takeof2};
class Runway {
    public:
        Runway(int limit);
        Runway(int limit,int max);
        Error_code can_land(const Plane &current);
        void can_land2(const Plane &current);
        Error_code can_depart(const Plane &current);
        Runway_activity activity1(int time, Plane &moving);// P1、P6 通用
        Runway_activity activity2(int time, Plane &moving, Plane &moving2);
        Runway_activity activity3(int time, Plane &moving, Plane &moving2);
```

```
        Runway_activity activity4(int time, Plane &moving, Plane &moving2, Plane
&moving3);
     Runway_activity activity5(int time, Plane &moving);
        void shut_down(int time) const;
    private:
        Extended_queue landing;
        Extended_queue takeoff;

        int queue_limit;
        int num_land_requests;
        int num_takeoff_requests;
        int num_landings;
        int num_takeoffs;
        int num_crash;
        int num_land_accepted;
        int num_takeoff_accepted;
        int num_land_refused;
        int num_takeoff_refused;
        int land_wait;
        int takeoff_wait;
        int idle_time;
};

#endif
```

Runway.cpp //共 302 行代码

```
#include"Runway.hpp"
#include<iostream>
using namespace std;

Runway::Runway(int limit):landing(limit),takeoff(limit){
   queue_limit = limit;
    num_land_requests = num_takeoff_requests = 0;
    num_landings = num_takeoffs = 0;
    num_land_refused = num_takeoff_refused = 0;
```

```
    num_land_accepted = num_takeoff_accepted = 0;
    land_wait = takeoff_wait = idle_time = 0;
    num_crash=-1;
}

Runway::Runway(int limit,int max):landing(max),takeoff(limit){
   queue_limit = limit;
    num_land_requests = num_takeoff_requests = 0;
    num_landings = num_takeoffs = 0;
    num_land_refused = num_takeoff_refused = 0;
    num_land_accepted = num_takeoff_accepted = 0;
    land_wait = takeoff_wait = idle_time = 0;
     num_crash=-1;
}

Error_code Runway::can_land(const Plane &current){
    Error_code result;
    if (landing.size() < queue_limit)
        result = landing.append(current);
    else
        result = fail;
    num_land_requests++;
    if (result != success)
        num_land_refused++;
    else
        num_land_accepted++;
    return result;
}

void Runway::can_land2(const Plane &current){
   landing.insert(current);
   num_land_requests++;
   num_land_accepted++;
}
```

```
Error_code Runway::can_depart(const Plane &current){
    Error_code result;
    if (takeoff.size() < queue_limit){
     result = takeoff.append(current);
    }else{
     result = fail;
    }

    num_takeoff_requests++;
    if (result != success)
        num_takeoff_refused++;
    else
        num_takeoff_accepted++;
    return result;
}
// P1、P6
Runway_activity Runway::activity1(int time, Plane &moving){
    Runway_activity in_progress;
    if (!landing.empty()) {
        landing.retrieve(moving);
        land_wait += time - moving.started();
        num_landings++;
        in_progress = land;
        landing.serve();
    }else if (!takeoff.empty()) {
        takeoff.retrieve(moving);
        takeoff_wait += time - moving.started();
        num_takeoffs++;
        in_progress = takeof;
        takeoff.serve();
    }else {
        idle_time++;
        in_progress = idle;
    }
    return in_progress;
```

```
}

// P2
Runway_activity Runway::activity2(int time, Plane &moving, Plane &moving2){
    Runway_activity in_progress;
    if(!landing.empty() and !takeoff.empty()){
        landing.retrieve(moving);
        land_wait += time - moving.started();
        num_landings++;
        landing.serve();
        takeoff.retrieve(moving2);
        takeoff_wait += time - moving2.started();
        num_takeoffs++;
        in_progress = land_and_takeof;
        takeoff.serve();
    }else if (!landing.empty()) {
        landing.retrieve(moving);
        land_wait += time - moving.started();
        num_landings++;
        in_progress = land;
        landing.serve();
    }else if (!takeoff.empty()) {
        takeoff.retrieve(moving2);
        takeoff_wait += time - moving2.started();
        num_takeoffs++;
        in_progress = takeof;
        takeoff.serve();
    }else {
        idle_time++;
        in_progress = idle;
    }
    return in_progress;
}

// P3
```

```
Runway_activity Runway::activity3(int time, Plane &moving, Plane &moving2){
    Runway_activity in_progress;
    if(!landing.empty() and !takeoff.empty()){
        landing.retrieve(moving);
        land_wait += time - moving.started();
        num_landings++;
        landing.serve();
        takeoff.retrieve(moving2);
        takeoff_wait += time - moving2.started();
        num_takeoffs++;
        in_progress = land_and_takeof;
        takeoff.serve();
    }else if (!landing.empty()) {
        if(landing.size() > 1){
            landing.retrieve(moving);
            land_wait += time - moving.started();
            num_landings++;
            landing.serve();
            landing.retrieve(moving2);
            land_wait += time - moving2.started();
            num_landings++;
            landing.serve();
            in_progress = land2;
        }else{
            landing.retrieve(moving);
            land_wait += time - moving.started();
            num_landings++;
            in_progress = land;
            landing.serve();
        }
    }else if (!takeoff.empty()) {
        if(takeoff.size() > 1){
            takeoff.retrieve(moving2);
            takeoff_wait += time - moving2.started();
            num_takeoffs++;
```

```
            takeoff.serve();
            takeoff.retrieve(moving);
            takeoff_wait += time - moving.started();
            num_takeoffs++;
            in_progress = takeof2;
            takeoff.serve();
        }else{
            takeoff.retrieve(moving2);
            takeoff_wait += time - moving2.started();
            num_takeoffs++;
            in_progress = takeof;
            takeoff.serve();
        }
    }else {
        idle_time++;
        in_progress = idle;
    }
    return in_progress;
}

// P4
Runway_activity Runway::activity4(int time, Plane &moving, Plane &moving2, Plane
&moving3){
    Runway_activity in_progress;
    if(!landing.empty() and !takeoff.empty()){
        if(landing.size() > 1){
            landing.retrieve(moving);
            land_wait += time - moving.started();
            num_landings++;
            landing.serve();
            landing.retrieve(moving2);
            land_wait += time - moving2.started();
            num_landings++;
            landing.serve();
            takeoff.retrieve(moving3);
```

```
            takeoff_wait += time - moving3.started();
            num_takeoffs++;
            in_progress = land2_and_takeof;
            takeoff.serve();
        }else if(takeoff.size() > 1){
            landing.retrieve(moving);
            land_wait += time - moving.started();
            num_landings++;
            landing.serve();
            takeoff.retrieve(moving2);
            takeoff_wait += time - moving2.started();
            num_takeoffs++;
            takeoff.serve();
            takeoff.retrieve(moving3);
            takeoff_wait += time - moving3.started();
            num_takeoffs++;
            takeoff.serve();
            in_progress = land_and_takeof2;
        }else{
            landing.retrieve(moving);
            land_wait += time - moving.started();
            num_landings++;
            landing.serve();
            takeoff.retrieve(moving2);
            takeoff_wait += time - moving2.started();
            num_takeoffs++;
            in_progress = land_and_takeof;
            takeoff.serve();
        }
    }else if (!landing.empty()) {
        if(landing.size() > 1){
            landing.retrieve(moving);
            land_wait += time - moving.started();
            num_landings++;
            landing.serve();
```

```
            landing.retrieve(moving2);
            land_wait += time - moving2.started();
            num_landings++;
            landing.serve();
            in_progress = land2;
        }else{
            landing.retrieve(moving);
            land_wait += time - moving.started();
            num_landings++;
            in_progress = land;
            landing.serve();
        }
    }else if (!takeoff.empty()) {
        if(takeoff.size() > 1){
            takeoff.retrieve(moving2);
            takeoff_wait += time - moving2.started();
            num_takeoffs++;
            takeoff.serve();
            takeoff.retrieve(moving);
            takeoff_wait += time - moving.started();
            num_takeoffs++;
            in_progress = takeof2;
            takeoff.serve();
        }else{
            takeoff.retrieve(moving2);
            takeoff_wait += time - moving2.started();
            num_takeoffs++;
            in_progress = takeof;
            takeoff.serve();
        }
    }else {
        idle_time++;
        in_progress = idle;
    }
    return in_progress;
```

```
}

// P5
Runway_activity Runway::activity5(int time, Plane &moving){
  Runway_activity in_progress;
  if( num_crash==-1) num_crash=0;
   num_crash +=landing.go(time);
     if (!landing.empty()) {
          landing.retrieve(moving);
          land_wait += time - moving.started();
          num_landings++;
          in_progress = land;
          landing.serve();
    }else if (!takeoff.empty()) {
          takeoff.retrieve(moving);
          takeoff_wait += time - moving.started();
          num_takeoffs++;
          takeoff.serve();
          in_progress = takeof;
    }else {
          idle_time++;
          in_progress = idle;
    }
    return in_progress;

}

void Runway::shut_down(int time) const{
  cout<< "\n模拟在 " << time << " 时间后结束。 " << endl
      <<"  处理的飞机数量:  "<<(num_land_requests+num_takeoff_requests)<<endl
      << "  要求着陆的飞机数量:  "<< num_land_requests << endl
      << "  要求起飞的飞机数量:  "<< num_takeoff_requests << endl
      << "  接受着陆的飞机数量:  "<< num_land_accepted << endl
      << "  接受起飞的飞机数量:  "<< num_takeoff_accepted << endl;
  if(num_crash==-1) cout<< "  拒绝着陆的飞机数量:  "<< num_land_refused << endl;
```

```
  else cout<<"  坠毁的飞机数量:  "<<num_crash<<endl;
  cout<< "  拒绝起飞的飞机数量:  "<< num_takeoff_refused << endl
      << "  着陆的飞机数量:  "<< num_landings << endl
      << "  起飞的飞机数量:  "<< num_takeoffs << endl;
  cout<< "  留在着陆队列里的飞机数量:  "<< landing.size() << endl
      << "  留在起飞队列里的飞机数量:  "<< takeoff.size() << endl;
  cout<< "  飞机跑道空闲的时间百分比:  "<< 100.0 * ((float) idle_time)/((float)
time) << "%" << endl;
  if(num_landings>0)  cout << "  飞机着陆的平均等待时长:  "<< ((float)
land_wait)/((float) num_landings) << endl;
  else  cout<<"  无飞机着陆"<<endl;
  if(num_takeoffs>0)  cout << "  飞机起飞的平均等待时长:  "<< ((float)
takeoff_wait)/((float) num_takeoffs) << endl;
  else  cout<<"  无飞机起飞"<<endl;
  cout << "  希望着陆的飞机平均被观察率为:  每单位时间 "<< ((float)
num_land_requests)/((float) time) << " 架。 " << endl;
  cout << "  希望起飞的飞机平均被观察率为:  每单位时间 "<< ((float)
num_takeoff_requests)/((float) time) << " 架。 " << endl;
  }
```

Plane.hpp //共 28 行代码

```
  #ifndef Plane_hpp
  #define Plane_hpp

  #include <cstdlib>
  using namespace std;

  enum Plane_status {null, arriving, departing};
  class Plane {
      public:
          Plane();
          Plane(int flt, int time, Plane_status status);
          void refuse() const;
          void wait()const;
          void land(int time) const;
```

```
        void fly(int time) const;
        void fall(int time)const;
        int started() const;
        int getgas()const;
        void decgas();
        void wait2()const;
    private:
        int flt_num;
        int clock_start;
        int gas;
        Plane_status state;
};

#endif
```

Plane.cpp //共 68 行代码

```
#include"Plane.hpp"
#include<iostream>
using namespace std;

Plane::Plane(int flt, int time, Plane_status status){
    flt_num = flt;
    clock_start = time;
    state = status;
    gas=rand()%5+5;
    cout << "   "<< flt_num <<" 号飞机"<< "准备";
    if (status == arriving)
        cout << "着陆,";
    else
        cout << "起飞,";
}
Plane::Plane(){
  gas=0;
    flt_num = -1;
```

```
    clock_start = -1;
    state = null;
}

void Plane::wait()const{
    if (state == arriving)
        cout << "进入着陆队列。 " << endl;
    else
        cout << "进入起飞队列。 " << endl;
}

void Plane::wait2()const{
  cout << "进入着陆队列,剩余 "<<gas<<" 单位燃料。" << endl;
}

void Plane::refuse() const{
    if (state == arriving)
     cout << "前往另一个机场着陆。 " << endl;
    else
        cout << "被告知稍后再尝试起飞。 " << endl;
}

void Plane::land(int time) const{
  int wait = time - clock_start;
    cout << time << ": 飞机编号 " << flt_num << " 在等待 "
        << wait << " 时间单元后着陆机场。 " << endl;
}

void Plane::fly(int time) const{
    int wait = time - clock_start;
    cout << time << ": 飞机编号 " << flt_num << " 在等待 "
        << wait << " 时间单元后从起飞队列起飞。 " << endl;
}

void Plane::fall(int time)const{
```

```
        cout << time << ": 飞机编号 " << flt_num << " 因油量不足坠毁。 " << endl;
    }

    int Plane::started() const{
        return clock_start;
    }

    int Plane::getgas()const{
      return gas;
    }

    void Plane::decgas(){
      gas--;
    }
```

Random.hpp //共 18 行代码

```
    #ifndef Random_hpp
    #define Random_hpp

    #include <cmath>
    #include <ctime>
    #include <cstdlib>
    #include <utility>
    using namespace std;

    class Random{
        public:
            Random(){}
            int poisson(float average);
            double U_Random();
            pair<int,int> set(int current_time);
    };

    #endif
```

Random.cpp //共 33 行代码

```
#include"Random.hpp"
#include<iostream>
using namespace std;

double Random::U_Random()
{
    double f;
    f = (float)(rand()%100);
    return f/100;
}

int Random::poisson(float average){
    int k = 0;
    long double p = 1.0;
    long double l = exp(-average);
    while (p>=l)
    {
        double u = U_Random();
        p *= u;
        k++;
    }
    return k-1;
}

pair<int,int> Random::set(int current_time){
    int land,fly;
    cout << "请输入 " << current_time << " 时刻请求着陆的飞机数量:  ";
    cin >> land;
    cout << "请输入 " << current_time << " 时刻请求起飞的飞机数量:  ";
    cin >> fly;
    pair<int,int> a(land,fly);
    return a;
}
```

main.cpp //共 27 行代码

```
#include<iostream>
#include<conio.h>
#include<windows.h>
#include"Function.hpp"
using namespace std;

//简化main函数，把功能函数声明及定义单独成文件
int main(){

  void (*Menu_Funp[6])(void);   // 起始页函数指针
  Menu_Funp[0]=Subpro1; Menu_Funp[1]=Subpro2; Menu_Funp[2]=Subpro3;
  Menu_Funp[3]=Subpro4; Menu_Funp[4]=Subpro5; Menu_Funp[5]=Subpro6;
  ME:
  Menu();  // 主菜单界面输出
  char inp;
  do{
      inp = _getch();
  }while(inp!='1'&&inp!='2'&&inp!='3'&&inp!='4'&&inp!='5'&&inp!='6'&&inp!='q');
  cout<<inp;
  if(inp=='q'){
      return 0;
  }
  system("cls");        // 清屏
  Menu_Funp[inp-'1']();  //实现相应功能
  system("cls");
  goto ME; // 实现循环功能，由用户决定退出程序
}
```