

Diagonal.h

```
4  #pragma once
5  #include <vector>
6
7  class DifferentSize : public std::exception{};
8  class InvalidIndex : public std::exception{};
9  class ReferenceToNullPart : public std::exception{};
10
11 class Diagonal{
12     private:
13         std::vector<int> vec;
14     public:
15         Diagonal(int n) : vec(n,0)
16         {
17
18         }
19
20         Diagonal(const std::vector<int> &v) : vec(v)
21         {
22
23         }
24
25         friend Diagonal operator+(const Diagonal &a, const Diagonal &b)
26         {
27             if(a.vec.size() != b.vec.size())
28             {
29                 throw DifferentSize();
30             }
31
32             Diagonal x(a.vec.size());
33
34             for(int i = 0; i < c.vec.size(); i++)
35             {
36                 c.vec[i] = a.vec[i] + b.vec[i];
37             }
38
39             return x;
40         }
41
42         friend Diagonal operator*(const Diagonal &a, const Diagonal &b)
43         {
44             if(a.vec.size() != b.vec.size())
45             {
46                 throw DifferentSize();
47             }
48
49             Diagonal x(a.vec.size());
50
51             for(int i = 0; i < c.vec.size(); i++)
52             {
53                 c.vec[i] = a.vec[i] * b.vec[i];
54             }
55
56             return x;
57         }
58
59
60         int get(int x, int y) const
61         {
62             if(x >= vec.size() || y >= vec.size() || x < 0 || y < 0)
63             {
64                 throw InvalidIndex();
65             }
66             else if(x == y)
67             {
68                 return vec[x];
69             }
70             else
71             {
72                 return 0;
73             }
74         }
75 }
```

```

76 void set(int x, int y, int n)
77 {
78     if(x >= vec.size() || y >= vec.size() || x < 0 || y < 0)
79     {
80         throw InvalidIndex();
81     }
82     else if(x == y)
83     {
84         vec[x] = n;
85     }
86     else
87     {
88         throw ReferenceToNullPart();
89     }
90 }
91
92 };

```

Cactus.cpp

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <sstream>
5
6  using namespace std;
7
8  struct Cactus{
9      std::string name, country, color;
10     int height;
11 };
12
13 enum Status{
14     abnorm, norm
15 };
16
17 bool read(std::ifstream &f, Cactus &c, Status &st)
18 {
19     std::string line;
20     if(getline(f,line))
21     {
22         std::istringstream sstream(line);
23         sstream >> c.name >> c.country >> c.color >> c.height
24         st = norm;
25     }
26     else
27     {
28         st = abnorm;
29     }
30
31     return st==norm;
32 }
33
34 int main(int argc, char* argv[])
35 {
36     std::ifstream in("in.txt");
37     //std::ifstream in(argv[1])
38     if(in.fail())
39     {
40         std::cout << "File open error" << std::endl;
41     }
42
43     Cactus c;
44     Status st;
45
46     std::ofstream out1("out1.txt");
47     std::ofstream out2("out2.txt");
48
49     while(read(in, c, st))
50     {
51         if(c.height > 10)
52         {
53             out1 << c.name << " " << c.country << std::endl;
54         }
55     }
56
57     return 0;
58 }

```

SeqInFile.hpp

```
1  #include <string>
2  #include <fstream>
3  #include <sstream>
4
5  class SeqInFile
6  {
7  private:
8      enum Status{
9          norm, abnorm
10     };
11
12     Status st;
13
14     std::ifstream inputStream;
15
16 public:
17     int num;
18     SeqInFile(std::string f)
19     {
20         inputStream.open(f);
21         if(inputStream.fail())
22         {
23             std::cout << "File open error!" << std::endl;
24         }
25     }
26
27     bool read()
28     {
29         if(inputStream >> num)
30         {
31             st = norm;
32         }
33         else
34         {
35             st = abnorm;
36         }
37
38         return st == norm;
39     }
40
41 };
42
```

Vasarlal.hpp + main.cpp

```
1  #include <string>
2  #include <fstream>
3  #include <sstream>
4
5  class SeqInFile
6  {
7  private:
8      enum Status{
9          norm, abnorm
10     };
11
12     Status st;
13
14     std::ifstream inputStream;
15
16 public:
17
18     struct Szamla{
19         std::string name;
20         int amount;
21     };
22     Szamla szamla;
23
```

```

24 SeqInFile(std::string f)
25 {
26     inputStream.open(f);
27     if(inputStream.fail())
28     {
29         std::cout << "File open error!" << std::endl;
30     }
31 }
32
33 bool read()
34 {
35     std::string line;
36
37     if(std::getline(inputStream,line);)
38     {
39         st = norm;
40         std::istringstream lineStream(line);
41         szamla.amount = 0;
42
43         lineStream >> szamla.name;
44
45         std::string termek;
46         int ar;
47         while(lineStream >> termek >> ar)
48         {
49             szamla.amount += ar;
50         }
51     }
52     else
53     {
54         st = abnorm;
55     }
56
57     return st == norm;
58 }
59
60 };
61

```

```

1  #include <iostream>
2  #include "SeqInFile.hpp"
3
4  using namespace std;
5
6  int main()
7  {
8      SeqInFile f("input2.txt");
9
10     f.read();
11
12     int bevetel = 0;
13     int maxSpending = f.szamla.amount;
14     std::string topSpender = f.szamla.name;
15
16     while(f.read())
17     {
18         bevetel += f.szamla.amount;
19
20         if(f.szamla.amount > maxSpending)
21         {
22             maxSpending = f.szamla.amount;
23             topSpender = f.szamla.name;
24         }
25     }
26
27     std::cout << bevetel << std::endl;
28
29     return 0;
30 }
31

```

mintazH.hpp + main.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <sstream>
4  #include <string>
5
6  class InvalidFileException : public std::exception{};
7
8  class SeqInFile
9  {
10 private:
11     enum Status{
12         norm, abnorm;
13     };
14
15     Status st;
16
17     std::ifstream beolvasas;
18
19     struct Korhaz{
20         std::string nev;
21         int betegDB;
22         int gepDB;
23     };
24
25     Korhaz korhaz;
26
27 public:
28     SeqInFile(std::string fileNev)
29     {
30         beolvasas.open(fileNev);
31         if(beolvasas.fail())
32         {
33             throw InvalidFileException();
34         }
35     }
36
37     struct NapiAdat{
38         std::string datum;
39         int fertozottSzam;
40         int korhazbanOSSZ;
41     };
42
43     NapiAdat napiadat;
44
45     bool Read()
46     {
47         std::string sor;
48
49         if(std::getline(beolvasas, sor))
50         {
51             st = norm;
52             std::istringstream adatok(sor);
53
54             adatok >> napiadat.datum >> napiadat.fertozottSzam;
55
56             while(adatok >> korhaz.nev >> korhaz.betegDB >> korhaz.gepDB)
57             {
58                 napiadat.korhazbanOSSZ += korhaz.betegDB;
59             }
60
61         }
62         else
63         {
64             st = abnorm;
65         }
66
67         return st == norm;
68     }
69 };
70
```

```

1  #include <iostream>
2  #include "SeqInFile.hpp"
3
4  using namespace std;
5
6  int main()
7  {
8      SeqInFile f("input.txt");
9
10     bool tobbM5k = false;
11     int maxKorhazban = 0;
12     std::string maxDatum;
13
14     while(f.Read())
15     {
16         tobbM5k = tobbM5k || f.napiadat.fertozottSzam > 5000;
17
18         if(f.korhaz.betegDB > maxKorhazban)
19         {
20             maxKorhazban = f.korhaz.betegDB;
21             maxDatum = f.napiadat.datum;
22         }
23     }
24
25     std::cout << tobbM5k << std::endl << maxDatum << std::endl;
26
27     SeqInFile f2("input.txt")
28
29     while(f2.Read() && f2.napiadat.korhazbanOSSZ <=100)
30     {
31     }
32
33     maxKorhazban = 0;
34     maxDatum = "";
35
36     int fertozottDB = 0;
37
38     while(f2.Read())
39     {
40         fertozottDB += f2.napiadat.fertozottSzam;
41         if(f2.napiadat.korhazbanOSSZ > maxKorhazban)
42         {
43             maxKorhazban = f2.napiadat.korhazbanOSSZ;
44             maxDatum = f2.napiadat.datum;
45         }
46     }
47
48     std::cout << fertozottDB << std::endl << maxDatum << " " << maxKorhazban;
49
50     return 0;
51 }
52
53

```

Product.hpp

```

1  #pragma once
2
3  #include <string>
4
5  class Product
6  {
7  private:
8      std::string name;
9      int price;
10 public:
11     Product(std::string n, int p): name(n), price(p) {}
12
13     std::string getName() {
14         return name;
15     }
16
17     int getPrice() {
18         return price;
19     }
20 };
21

```

Department.hpp

```
1  #pragma once
2
3  #include "Product.hpp"
4  #include <vector>
5  #include <fstream>
6
7  class Department
8  {
9  public:
10     std::vector<Product*> stock;
11
12     Department(std::string fileName)
13     {
14         std::ifstream f(fileName);
15         std::string name;
16         int price;
17
18         while(f>>name>>price)
19         {
20             stock.push_back(new Product(name,price));
21         }
22     }
23
24     void takeOutFromStock(Product *product)
25     {
26         bool productFound = false;
27         int index = 0;
28
29         while(index < stock.size() && !productFound)
30         {
31             if(stock[index] == product)
32             {
33                 productFound = true;
34             }
35             index++;
36         }
37         index--;
38
39         if(productFound)
40         {
41             stock.erase(stock.begin()+index);
42         }
43     }
44 }
45
```

Store.hpp

```
1  #pragma once
2
3  #include "Department.hpp"
4
5  class Store
6  {
7  public:
8     Department *foods;
9     Department *technical;
10
11     Store(std::string foodsFile, std::string technicalFile)
12     {
13         foods = new Department(foodsFile);
14         technical = new Department(technicalFile);
15     }
16
17     ~Store()
18     {
19         delete foods;
20         delete technical;
21     }
22 };
23
```

Costumer.hpp

```
1  #pragma once
2
3  include "Store.hpp"
4  include <fstream>
5  include <iostream>
6  include <string>
7
8  class Costumer{
9
10 private:
11     std::vector<std::string> list;
12
13     void buy(Product *p, Department *d)
14     {
15         d->takeOutFromStock(p);
16         std::cout << p->getName() << " " << p->getPrice() << std::endl;
17     }
18
19 public:
20     Costumer(std::string fileName)
21     {
22         std::ifstream f(fileName);
23
24         std::string s;
25         while(f >> s){
26             list.push_back(s);
27         }
28     }
29
30     void goShopping(Store &st)
31     {
32         for(std::string productName : list)
33         {
34             Product *product;
35             if(search(productName, s.foods, product))
36             {
37                 buy(product, s.foods);
38             }
39             if(minsearch(productName, s.technical, product))
40             {
41                 buy(product, s.technical);
42             }
43         }
44     }
45
46     bool search(std::string name, Department *d, Product* &p)
47     {
48         bool l = false;
49
50         for(Product *p : d->stock)
51         {
52             if(name == p->getName())
53             {
54                 l = true;
55                 product = p;
56                 break;
57             }
58         }
59
60         return l;
61     }
```



```

63     bool minsearch(const std::string &name, Department* d, Product* &Product) const
64     {
65         bool l = false;
66         int MIN;
67
68         for(Product *p : d->stock)
69         {
70             if(p->getName() != name) continue;
71             if(l)
72             {
73                 if(MIN > p->getPrice())
74                 {
75                     MIN = p->getPrice();
76                     product = p;
77                 }
78             }
79             else{
80                 l = true;
81                 MIN = p->getPrice();
82                 product = p;
83             }
84         }
85
86         return l;
87     }
88 };
89

```

main.cpp

```

1  #include "Costumer.hpp"
2  #include "Store.hpp"
3
4  using namespace std;
5
6  int main()
7  {
8
9      Costumer c("costumer.txt");
10     Store s("food.txt", "technical.txt");
11
12     c.goShopping(s);
13
14     return 0;
15 }
16

```

Plants.hpp

```

1  #pragma once
2
3  class Plant{
4  protected:
5      int ripeningTime;
6      Plant(int r) : ripeningTime(r) {}
7
8  public:
9      virtual bool isVegetable() const {return false;}
10     virtual bool isFlower() const {return false;}
11     int getRipeningTime() const {return ripeningTime;}
12
13     virtual ~Plant() {}
14
15 };
16

```

```

17 class Vegetable : public Plant{
18     protected:
19         Vegetable(int r) : Plant(r) {}
20
21     public:
22         bool isVegetable() const override {return false;}
23 };
24
25 class Flower : public Plant{
26     protected:
27         Flower(int r) : Plant(r) {}
28
29     public:
30         bool isFlower() const override {return false;}
31 };
32
33 class Potato : public Vegetable{
34     private:
35         static Potato* _inst;
36         Potato() : Vegetable(5) {}
37
38     public:
39         static Potato* inst();
40         static void destroy();
41 };
42
43 class Pea : public Vegetable{
44     private:
45         static Pea* _inst;
46         Pea() : Vegetable(3) {}
47
48     public:
49         static Pea* inst();
50         static void destroy();
51 };
52
53 class Onion : public Vegetable{
54     private:
55         static Onion* _inst;
56         Onion() : Vegetable(4) {}
57
58     public:
59         static Onion* inst();
60         static void destroy();
61 };
62
63 class Rose : public Flower{
64     private:
65         static Rose* _inst;
66         Rose() : Vegetable(8) {}
67
68     public:
69         static Rose* inst();
70         static void destroy();
71 };
72
73 class Carnation : public Flower{
74     private:
75         static Carnation* _inst;
76         Carnation() : Vegetable(10) {}
77
78     public:
79         static Carnation* inst();
80         static void destroy();
81 };
82
83 class Tulip : public Flower{
84     private:
85         static Tulip* _inst;
86         Tulip() : Vegetable(7) {}
87
88     public:
89         static Tulip* inst();
90         static void destroy();
91 };

```

Plants.cpp

```
1  #include "plants.hpp"
2
3  Potato* Potato::_inst = nullptr;
4  Potato* Potato::inst(){
5      if(_inst == nullptr){
6          _inst = new Potato();
7      }
8      return _inst;
9  }
10
11 void Potato::destroy()
12 {
13     if(_inst != nullptr){
14         delete _inst;
15         _inst = nullptr;
16     }
17 }
18
19 Pea* Pea::_inst = nullptr;
20 Pea* Pea::inst(){
21     if(_inst == nullptr){
22         _inst = new Pea();
23     }
24     return _inst;
25 }
26
27 void Pea::destroy()
28 {
29     if(_inst != nullptr){
30         delete _inst;
31         _inst = nullptr;
32     }
33 }
34
35 Onion* Onion::_inst = nullptr;
36 Onion* Onion::inst(){
37     if(_inst == nullptr){
38         _inst = new Onion();
39     }
40     return _inst;
41 }
42
43 void Onion::destroy()
44 {
45     if(_inst != nullptr){
46         delete _inst;
47         _inst = nullptr;
48     }
49 }
50
51 Rose* Rose::_inst = nullptr;
52 Rose* Rose::inst(){
53     if(_inst == nullptr){
54         _inst = new Rose();
55     }
56     return _inst;
57 }
58
59 void Rose::destroy()
60 {
61     if(_inst != nullptr){
62         delete _inst;
63         _inst = nullptr;
64     }
65 }
66
67 Carnation* Carnation::_inst = nullptr;
68 Carnation* Carnation::inst(){
69     if(_inst == nullptr){
70         _inst = new Carnation();
71     }
72     return _inst;
73 }
74
```

```

75 void Carnation::destroy()
76 {
77     if(_inst != nullptr){
78         delete _inst;
79         _inst = nullptr;
80     }
81 }
82
83 Tulip* Tulip::_inst = nullptr;
84 Tulip* Tulip::inst(){
85     if(_inst == nullptr){
86         _inst = new Tulip();
87     }
88     return _inst;
89 }
90
91 void Tulip::destroy()
92 {
93     if(_inst != nullptr){
94         delete _inst;
95         _inst = nullptr;
96     }
97 }
98

```

Parcel.hpp

```

1  #pragma once
2
3  #include "plants.hpp"
4  #include <iostream>
5
6  class Parcel{
7  private:
8      Plant* _p;
9      int plantingDate;
10
11  public:
12      Parcel() : _p(nullptr), plantingDate(0) {}
13
14      void plant(Plant *p, int date){
15          if(_p == nullptr){
16              _p = p;
17              plantingDate = date;
18          } else{
19              std::cout << "Plant already planted here!" << std::endl;
20          }
21      }
22
23      void harvest() {_p = nullptr;}
24
25      bool isRipe(int month){
26          if(_p != nullptr && (month - plantingDate) == _p->getRipeningTime()){
27              return true;
28          }
29          return false;
30      }
31  };
32

```

Garden.hpp

```
1  #pragma once
2
3  #include <vector>
4  #include "parcel.hpp"
5
6  class Garden{
7  private:
8      std::vector<Parcel*> parcels;
9
10 public:
11     Garden(int n){
12         if(n>=1){
13             parcels.resize(n);
14
15             for(int i=0; i<n; i++){
16                 parcels[i] = new Parcel();
17             }
18         }
19     }
20
21     std::vector<int> canHarvest(int date){
22         std::vector<int> result;
23
24         for(int i=0; i<parcels.size(); i++){
25             if(parcels[i]->isRipe(date)){
26                 result.push_back(i);
27             }
28         }
29         return result;
30     }
31
32     Parcel* getParcel(int i) const{
33         return parcels[i];
34     }
35
36     ~Garden(){
37         for(Parcel* p: parcels){
38             delete p;
39         }
40     }
41 };
42
```

Gardner.hpp

```
1  #pragma once
2
3  #include "garden.hpp"
4
5  class Gardner{
6  public:
7      Garden* _g;
8
9      Gardner(Garden g) : _g(g) {}
10
11     void harvest(int i){
12         _g->getParcel(i)->harvest();
13     }
14
15     void plant(int i, Plant* p, int date){
16         _g->getParcel(i)->plant(p, date);
17     }
18 };
19
```

main.cpp

```
1  #include <iostream>
2  #include "gardner.hpp"
3  #include "plants.hpp"
4
5  using namespace std;
6
7  int main()
8  {
9      Garden* garden = new Garden(5);
10     Gardner* gardner = new Gardner(garden);
11
12     gardner->plant(1, Potato::inst(), 3);
13     gardner->plant(2, Pea::inst(), 3);
14     gardner->plant(4, Rose::inst(), 3);
15
16     std::cout << "A betakarítható parcellák azonosítói: ";
17     for(int i=0 : gardner->_g->canHarvest(6)){
18         std::cout << i << " ";
19     }
20     std::cout << std::endl;
21
22     delete gardner;
23     delete garden;
24
25     Potato::destroy;
26     Pea::destroy;
27     Rose::destroy;
28
29     return 0;
30 }
31
```

Animal.hpp

```
1  #pragma once
2
3  class Animal{
4  public:
5      Animal(double w, bool l) : _weight(w), _male(l) {}
6
7      virtual bool isLion() const {return false;}
8      virtual bool isRhino() const {return false;}
9      virtual bool isElephant() const {return false;}
10     virtual ~Animal(){}
11
12     double _weight;
13     bool _male;
14 };
15
16 class Lion : public Animal{
17 public:
18     Lion(double w, bool l) : Animal(w,l) {}
19     bool isLion const override {return true;}
20 };
21
22 class Rhino : public Animal{
23 public:
24     Rhino(double w, bool l, double h) : Animal(w,l), _horn(h) {}
25     bool isRhino const override {return true;}
26
27     double _horn;
28 };
29
30 class Elephant : public Animal{
31 public:
32     Elephant(double w, bool l, double lt, double rt) : Animal(w,l), _leftTusk(lt), _rightTusk(rt) {}
33     bool isElephant const override {return true;}
34
35     double _leftTusk, _rightTusk;
36 };
37
```

Trophy.hpp

```
1 #pragma once
2 #include <string>
3 #include "Animal.hpp"
4
5 class Trophy{
6 public:
7     std::string place;
8     std::string date;
9     Animal* animal;
10
11     Trophy(Animal* a, std::string p, std::string d) : animal(a), place(p), date(d) {}
12
13     ~Trophy(){
14         if(animal != nullptr){
15             delete animal;
16         }
17     }
18 };
19
```

Hunter.hpp

```
1 #pragma once
2
3 #include "Trophy.hpp"
4 #include <vector>
5
6 class Hunter{
7 public:
8     std::string name;
9     int age;
10     std::vector<Trophy*> trophies;
11
12     Hunter(std::string n, int a): name(n), age(a) {}
13
14     void capture(Animal* a, std::string p, std::string d);
15     int countMaleLions() const;
16     float maxHornWeightRatio() const;
17     bool searchEqualTusks() const;
18
19     ~Hunter();
20 };
21
```

Hunter.cpp

```
1 #include "Hunter.hpp"
2
3 void Hunter::capture(Animal* a, std::string p, std::string d){
4     trophies.push_back(new Trophy(a,p,d))
5 }
6
7 int Hunter::countMaleLions() const{
8     int count = 0;
9     for(Trophy* t : trophies){
10         if(t->animal->isLion() && t->animal->_male){
11             count++;
12         }
13     }
14     return count;
15 }
```

```

17 float Hunter::maxHornWeightRatio() const{
18     float maxRate = -1;
19     float rate;
20
21     for(Trophy* t: trophies){
22         if(t->animal->isRhino){
23             Rhino* rhino = (Rhino*)t->animal;
24             rate = rhino->_horn/rhino->_weight;
25             if(rate > maxRate){
26                 maxRate = rate;
27             }
28         }
29     }
30
31     return maxRate;
32 }
33
34 bool Hunter::searchEqualTusks() const{
35     bool l = false;
36     for(Trophy* t: trophies){
37         if(t->animal->isElephant()){
38             Elephant* e = (Elephant*)t->animal;
39             l = e->_leftTusk == e->_rightTusk;
40         }
41     }
42
43     if(l){
44         break;
45     }
46
47     return l;
48 }
49
50 ~Hunter(){
51     for(Trophy* t: trophies){
52         delete t;
53     }
54 }
55

```

main.cpp

```

1  #include <iostream>
2  #include <fstream>
3  #include <sstream>
4
5  #include "Hunter.hpp"
6
7  void read(Hunter &h, std::string fileName)
8  {
9      std::ifstream f(fileName);
10     std::string line, place, date, species, gender;
11     double weight, lTusk, rTusk, horn;
12     Animal* animal;
13
14     while(getline(f,line)){
15         std::istringstream stringStream(line);
16
17         stringStream >> place >> date >> species >> weight >> gender;
18
19         if(species == "lion"){
20             animal = new Lion(weight, gender == "male");
21         } else if(species == "rhino"){
22             stringStream >> horn;
23             animal = new Rhino(weight,gender == "male", horn)
24         } else if(species == "elephant"){
25             stringStream >> lTusk >> rTusk;
26             animal = new Elephant(weight, gender == "male", lTusk, rTusk);
27         }
28
29         h.capture(animal,place,date);
30     }
31 }
32
33 int main()
34 {
35     Hunter hunter("Bill", 55);
36     read(hunter, "input.txt");
37
38     std::cout << hunter.countMaleLions() << std::endl;
39     std::cout << hunter.maxHornWeightRatio() << std::endl;
40     std::cout << hunter.searchEqualTusks() << std::endl;
41
42     return 0;
43 }
44
45

```


SolarSystem.hpp

```
1  #pragma once
2
3  #include <vector>
4
5  class Planet;
6  class StarShip;
7
8  class SolarSystem{
9  private:
10     SolarSystem(){}
11     static SolarSystem* _instance;
12
13 public:
14     std::vector<Planet*> planets;
15
16     static SolarSystem* Instance() {
17         if(_instance == nullptr){
18             _instance = new SolarSystem();
19         }
20         return _instance;
21     }
22
23     static void destroy(){
24         if(_instance != nullptr){
25             delete _instance;
26         }
27     }
28
29     ~SolarSystem();
30
31     bool MaxFirepower(StarShip* &bestShip) const;
32     Planet* Defenseless() const;
33 };
34
```

SolarSystem.cpp

```
1  #include "SolarSystem.hpp"
2  #include "Planet.hpp"
3
4  SolarSystem* SolarSystem::_instance = nullptr;
5
6  bool SolarSystem::MaxFirepower(StarShip* &bestShip) const{
7      bool l = false;
8      int maxFp;
9
10     for(Planet* p:planets){
11         int power;
12         StarShip* ship;
13
14         bool l1 = p->MaxFirepower(power,ship);
15
16         if(!l1){
17             continue;
18         } else if(!l){
19             l = true;
20             maxFp = power;
21             bestShip = ship;
22         } else if(power > maxFp){
23             maxFp = power;
24             bestShip = ship;
25         }
26         return l;
27     }
28 }
29
30 Planet* SolarSystem::Defenseless() const{
31     for(Planet* p:planets){
32         if(p->ships.size() == 0){
33             return p;
34         }
35     }
36
37     return nullptr;
38 }
39
40 SolarSystem::~SolarSystem(){
41     for(Planet* p : planets){
42         delete p;
43     }
44 }
45
```

Planet.hpp

```
1  #pragma once
2
3  #include <string>
4  #include <vector>
5
6  class Starship;
7
8  class Planet{
9  public:
10     std::string name;
11     std::vector<StarShip*> ships;
12
13     Planet(std::string n) : name(n){}
14     bool MaxFirepower(int &maxFp, StarShip* &ship) const;
15     int TotalShields() const;
16
17     ~Planet();
18 };
19
```

Planet.cpp

```
1  #include "Planet.hpp"
2  #include "StarShip.hpp"
3
4  bool Planet::MaxFirepower(int &maxFp, StarShip* &ship) const{
5      bool l = false;
6
7      for(StarShip* ss : ships){
8          int fp = ss->Firepower();
9          if(!l){
10             l = true;
11             maxFp = fp;
12             ship = ss;
13          } else if(fp > maxFp){
14             maxFp = fp;
15             ship = ss;
16          }
17      }
18
19      return l;
20  }
21
22  int Planet::TotalShields() const{
23      int sum = 0;
24      for(StarShip* ss : ships){
25          sum += ss->_shield;
26      }
27      return sum;
28  }
29
30  Planet::~~Planet(){
31      for(StarShip *ss : ships){
32          delete ss;
33      }
34  }
35
```

Starship.hpp

```
1 #pragma once
2
3 #include "Planet.hpp"
4
5 class StarShip{
6 public:
7     std::string _name;
8     int _shield;
9     int _armor;
10    int _spaceguard;
11    Planet* _planet;
12
13    StarShip(std::string name, int shield, int armor, int guard) : _name(name), _shield(shield), _armor(armor), _spaceguard(guard) {}
14    virtual ~StarShip(){}
15
16    void Protect(Planet* p){
17        _planet = p;
18        p->ships.push_back(this);
19    }
20
21    virtual int Firepower() const = 0;
22 };
23
24 class Breaker : public StarShip
25 {
26 public:
27     Breaker(std::string name, int shield, int armor, int guard) : StarShip(name, shield, armor, guard) {}
28     int Firepower() const override {return _armor/2;}
29 };
30
31 class Lander : public StarShip
32 {
33 public:
34     Breaker(std::string name, int shield, int armor, int guard) : StarShip(name, shield, armor, guard) {}
35     int Firepower() const override {return _spaceguard;}
36 };
37
38 class Laser : public StarShip
39 {
40 public:
41     Breaker(std::string name, int shield, int armor, int guard) : StarShip(name, shield, armor, guard) {}
42     int Firepower() const override {return _shield;}
43 };
44
```

main.cpp

```
1 #include <iostream>
2 #include <fstream>
3
4 #include "SolarSystem.hpp"
5 #include "Planet.hpp"
6 #include "StarShip.hpp"
7
8 using namespace std;
9
10 int main()
11 {
12     SolarSystem* ss = SolarSystem::Instance();
13
14     std::ifstream f("input.txt");
15
16     std::string pName, sName, sType;
17     int shield, armor, guard;
18     int n, m;
19
20     f >> n;
21
22     for(int i=0; i<n; i++){
23         f >> pName >> m;
24
25         Planet* p = new Planet(pName);
26
27         for(int j = 0; j<m; j++){
28             f >> sName >> sType >> shield >> armor >> guard;
29             StarShip* sh;
```

```

30
31
32         if(sType == "Breaker"){
33             sh = new Breaker(sName, shield,armor,guard);
34         } else if(sType == "Lander"){
35             sh = new Lander(sName, shield,armor,guard);
36         } else if(sType == "Laser"){
37             sh = new Laser(sName, shield,armor,guard);
38         }
39
40         sh->Protect(p);
41     }
42
43     ss->planets.push_back(p);
44 }
45
46 StarShip* ship;
47 if(ss->MaxFirepower(ship)){
48     std::cout << ship->_name << std::endl;
49 }
50
51 for(Planet* p:ss->planets){
52     if(p == "Earth"){
53         std::cout << p->TotalShields() << std::endl;
54     }
55 }
56
57 std::cout << ss->Defenseless()->name << " is defenseless!" << std::endl;
58 }

```

Signal.hpp

```

1  #pragma once
2
3  enum Signal{open, close, start, stop, press};
4

```

Magnetron.hpp

```

1  #pragma once
2
3  #include "Signal.hpp"
4
5  class Microwave;
6
7  enum MagnetronState {on, off};
8
9  class Magnetron{
10
11     Microwave* micro;
12
13     MagnetronState = off;
14
15 public:
16     Magnetron(Microwave* m) : micro(m) {}
17     void send(Signal sig);
18
19 }
20

```

Magnetron.cpp

```
1  #include "Magnetron.hpp"
2  #include "Signal.hpp"
3  #include "Microwave.hpp"
4
5  void Magnetron::send(Signal sig){
6      if(state == on){
7          switch(sig){
8              case press:
9                  state = off;
10                 micro->lamp->send(Signal stop);
11                 break;
12             case open:
13                 state = off;
14                 break;
15             default:
16                 break;
17         }
18     } else if(state == off){
19         switch(sig){
20             case press:
21                 state = on;
22                 micro->lamp->send(Signal start);
23                 break;
24             default:
25                 break;
26         }
27     }
28 }
29
```

Lamp.hpp

```
1  #pragma once
2
3  #include "Signal.hpp"
4
5  class Microwave;
6
7  enum LampState {lit, notLit};
8
9  class Lamp{
10
11     Microwave* micro;
12
13     LampState state = notLit;
14
15 public:
16     Lamp(Microwave* m) : micro(m) {}
17     void send(Signal sig);
18
19 };
20
```

Lamp.cpp

```
1 #include "Lamp.hpp"
2 #include "Signal.hpp"
3 #include "Microwave.hpp"
4
5 void Lamp::send(Signal sig){
6     switch(sig){
7         case open:
8             state = lit;
9             break;
10        case close:
11            state = notLit;
12            break;
13        case start:
14            state = lit;
15            break;
16        case stop:
17            state = notLit;
18            break;
19        default:
20            break;
21    }
22 }
23
```

Door.hpp

```
1 #pragma once
2
3 #include "Signal.hpp"
4
5 class Microwave;
6
7 enum DoorState {opened, closed};
8
9 class Door{
10
11     Microwave* micro;
12
13     DoorState state = closed;
14
15 public:
16     Door(Microwave* m) : micro(m) {}
17     void open();
18     void close();
19 }
20
```

Door.cpp

```
1 #include "Button.hpp"
2 #include "Signal.hpp"
3 #include "Microwave.hpp"
4
5 void Button::press() {
6     micro->magnetron->send(Signal::press)
7 }
8
```

Button.hpp

```
1 #pragma once
2
3 #include "Signal.hpp"
4
5 class Microwave;
6
7 class Button{
8
9     Microwave* micro;
10
11 public:
12     Button(Microwave* m) : micro(m) {}
13
14     void press();
15
16 }
17
```

Button.cpp

```
1 #include "Button.hpp"
2 #include "Signal.hpp"
3 #include "Microwave.hpp"
4
5 void Button::press() {
6     micro->magnetron->send(Signal::press)
7 }
8
```

Microwave.hpp

```
1 #pragma once
2
3 #include "Magnetron.hpp"
4 #include "Lamp.hpp"
5 #include "Door.hpp"
6 #include "Button.hpp"
7
8 class Microwave{
9 public:
10     Magnetron* magnetron;
11     Lamp* lamp;
12     Door* door;
13     Button* button;
14
15     Microwave() {
16         magnetron = new Magnetron(this);
17         lamp = new Lamp(this);
18         door = new Door(this);
19         button = new Button(this);
20     }
21
22     ~Microwave() {
23         delete magnetron;
24         delete lamp;
25         delete door;
26         delete button;
27     }
28
29 };
30
31
```

Vaccine.hpp

```
1  #pragma once
2
3  #include <string>
4
5  class Vaccine
6  {
7  private:
8      /* data */
9  public:
10     int expirationDate;
11     Vaccine(int d): expirationDate(d) {}
12
13     virtual std::string Name() = 0;
14     virtual int Repetition() = 0;
15
16     virtual ~Vaccine() {}
17 };
18
19 class Pfizer : public Vaccine
20 {
21 public:
22
23     std::string Name() override{
24         return "Pfizer";
25     }
26
27     int Repetition() override{
28         return 28;
29     }
30
31     Pfizer(int d): Vaccine(d) {}
32 };
33
34 class Astra : public Vaccine
35 {
36 public:
37
38     std::string Name() override{
39         return "Astra";
40     }
41
42     int Repetition() override{
43         return 21;
44     }
45
46     Astra(int d): Vaccine(d) {}
47 };
48
49 class Moderna : public Vaccine
50 {
51 public:
52
53     std::string Name() override{
54         return "Moderna";
55     }
56
57     int Repetition() override{
58         return 84;
59     }
60
61     Moderna(int d): Vaccine(d) {}
62 };
```


Vaccination.hpp

```
1  #pragma once
2
3  #include "Vaccine.hpp"
4
5  class Vaccination
6  {
7  private:
8      /* data */
9  public:
10     int date;
11     Vaccine* vaccine;
12 };
13
14
```

Patient.hpp

```
1  #pragma once
2
3  #include<vector>
4  #include "Vaccination.hpp"
5
6  class Patient
7  {
8  private:
9      /* data */
10 public:
11     std::vector<Vaccination*> vaccinations;
12     std::string TAJ;
13
14     int NumOfVacc()
15     {
16         return vaccinations.size();
17     }
18     Patient(std::string t): TAJ(t) {}
19     ~Patient();
20 };
21
```

Hospital.hpp

```
1  #pragma once
2
3  #include <iostream>
4
5  #include "Vaccine.hpp"
6  #include "Patient.hpp"
7
8  class Hospital
9  {
10 public:
11     std::string name;
12     std::vector<Vaccine*> vaccines;
13     std::vector<Patient*> registered;
14     Hospital();
15
16     void Procure2(Vaccine* v)
17     {
18         vaccines.push_back(v);
19     }
20
21
```

```

20
21 void Procure(std::string vName)
22 {
23     if(vName == "Pfizer")
24     {
25         std::cout << "Vaccine Procured" << std::endl;
26         vaccines.push_back(new Pfizer(10));
27     }
28     else if(vName == "Astra")
29     {
30         std::cout << "Vaccine Procured" << std::endl;
31         vaccines.push_back(new Astra(10));
32     }
33     else if(vName == "Moderna")
34     {
35         std::cout << "Vaccine Procured" << std::endl;
36         vaccines.push_back(new Moderna(10));
37     }
38     else
39     {
40         std::cout << "Vaccine cannot be procured" << std::endl;
41     }
42 }
43 void Register(Patient* p)
44 {
45     bool l = false;
46     for(Patient* e : registered)
47     {
48         if(e->TAJ == p->TAJ)
49         {
50             l = true;
51             std::cout << "Patient already registered" << std::endl;
52             break;
53         }
54     }
55     if(!l)
56     {
57         std::cout << "Patient registered" << std::endl;
58         registered.push_back(p);
59     }
60 }
61 void Vaccinate(Patient* p, std::string vName/*, int date*/)
62 {
63     bool l1 = false;
64     bool l2 = false;
65
66     Vaccine* vaccine;
67
68     for(Vaccine* e : vaccines)
69     {
70         if(e->Name() == vName /*&& e->expirationDate >= date*/)
71         {
72             l1 = true;
73             vaccine = e;
74             break;
75         }
76     }
77
78     for(Patient* e : registered)
79     {
80         if(e->TAJ == p->TAJ)
81         {
82             l2 = true;
83             break;
84         }
85     }

```

```

87         if(l1 && l2)
88         {
89             //vaccines.erase(std::remove(vaccines.begin(), vaccines.end(), vaccine), vaccines.end());
90             for(int i = 0; i<vaccines.size();i++)
91             {
92                 if(vaccines[i] == vaccine)
93                 {
94                     vaccines[i] = vaccines.back();
95                     vaccines.pop_back();
96                     //break;
97                 }
98             }
99
100             Vaccination* v = new Vaccination();
101             v->vaccine = vaccine;
102             p->vaccinations.push_back(v);
103             std::cout << "Patient vaccinated" << std::endl;
104         }
105         else
106         {
107             std::cout << "Vaccine not found or Patient not registered" << std::endl;
108         }
109     }
110
111     int NumOfMultiple()
112     {
113         int count = 0;
114         for(Patient* e : registered)
115         {
116             if(e->NumOfVacc() >= 2)
117             {
118                 count++;
119             }
120         }
121
122         return count;
123     }
124     ~Hospital();
125 };

```

main.cpp

```

1  #include<iostream>
2  #include "Hospital.hpp"
3
4  int main()
5  {
6      Hospital* h = new Hospital();
7      Patient* p1 = new Patient("1");
8      Patient* p2 = new Patient("2");
9      Patient* p3 = new Patient("3");
10
11      h->Procure("Pfizer");
12      h->Procure("Pfizer");
13      h->Procure("Pfizer");
14      h->Procure("Pfizersd");
15
16      h->Register(p1);
17      h->Register(p2);
18      h->Register(p2);
19
20      h->Vaccinate(p1,"Pfizer");
21      h->Vaccinate(p2,"Pfizer");
22      h->Vaccinate(p2,"Pfizer");
23      h->Vaccinate(p2,"Pfizer");
24      h->Vaccinate(p1,"Astra");
25
26      std::cout << h->NumOfMultiple() << std::endl;
27      return 0;
28  }

```

Ground.hpp

```
5  #pragma once
6
7  #include <string>
8
9  class Greenfinch;
10 class DuneBeetle;
11 class Squelchy;
12
13 // class of abstract grounds
14 class Ground{
15 public:
16     virtual Ground* change(Greenfinch *p) = 0;
17     virtual Ground* change(DuneBeetle *p) = 0;
18     virtual Ground* change(Squelchy *p) = 0;
19     virtual ~Ground() {}
20     static Ground* create(int k);
21 };
22
23 // class of sand
24 class Sand : public Ground
25 {
26 public:
27     static Sand* instance();
28     Ground* change(Greenfinch *p) override;
29     Ground* change(DuneBeetle *p) override;
30     Ground* change(Squelchy *p) override;
31     void static destroy() { if ( nullptr!=_instance ) delete _instance; _instance = nullptr; }
32 private:
33     Sand(){}
34     static Sand* _instance;
35 };
36
37 // class of grass
38 class Grass : public Ground
39 {
40 public:
41     static Grass* instance();
42     Ground* change(Greenfinch *p) override;
43     Ground* change(DuneBeetle *p) override;
44     Ground* change(Squelchy *p) override;
45     static void destroy() { if ( nullptr!=_instance ) delete _instance; _instance = nullptr; }
46 private:
47     Grass(){}
48     static Grass* _instance;
49 };
50
51 // class of marsh
52 class Marsh : public Ground
53 {
54 public:
55     static Marsh* instance();
56     Ground* change(Greenfinch *p) override;
57     Ground* change(DuneBeetle *p) override;
58     Ground* change(Squelchy *p) override;
59     void static destroy() { if ( nullptr!=_instance ) delete _instance; _instance = nullptr; }
60 private:
61     Marsh(){}
62     static Marsh* _instance;
63 };
64
```

Ground.cpp

```
5  #include "ground.h"
6  #include "creature.h"
7
8  using namespace std;
9
10 Ground* Ground::create(int k)
11 {
12     switch (k)
13     {
14         case 0: return Sand::instance();
15         case 1: return Grass::instance();
16         case 2: return Marsh::instance();
17     }
18     return nullptr;
19 }
```

```

21 // implementation of class Sand
22 Sand* Sand::_instance = nullptr;
23 Sand* Sand::instance()
24 {
25     if(_instance == nullptr) {
26         _instance = new Sand();
27     }
28     return _instance;
29 }
30
31 Ground* Sand::change(Greenfinch *p)
32 {
33     p->changePower(-2);
34     return this;
35 }
36
37 Ground* Sand::change(DuneBeetle *p)
38 {
39     p->changePower(3);
40     return this;
41 }
42
43 Ground* Sand::change(Squelchy *p)
44 {
45     p->changePower(-5);
46     return this;
47 }
48
49 // implementation of class Grass
50 Grass* Grass::_instance = nullptr;
51 Grass* Grass::instance()
52 {
53     if(_instance == nullptr) {
54         _instance = new Grass();
55     }
56     return _instance;
57 }
58
59 Ground* Grass::change(Greenfinch *p)
60 {
61     p->changePower(1);
62     return this;
63 }
64
65 Ground* Grass::change(DuneBeetle *p)
66 {
67     p->changePower(-2);
68     return Sand::instance();
69 }
70
71 Ground* Grass::change(Squelchy *p)
72 {
73     p->changePower(-2);
74     return Marsh::instance();
75 }
76

```

```

77 // implementation of class Marsh
78 Marsh* Marsh::_instance = nullptr;
79 Marsh* Marsh::instance()
80 {
81     if(_instance == nullptr) {
82         _instance = new Marsh();
83     }
84     return _instance;
85 }
86
87 Ground* Marsh::change(Greenfinch *p)
88 {
89     p->changePower(-1);
90     return Grass::instance();
91 }
92
93 Ground* Marsh::change(DuneBeetle *p)
94 {
95     p->changePower(-4);
96     return Grass::instance();
97 }
98
99 Ground* Marsh::change(Squelchy *p)
100 {
101     p->changePower(6);
102     return this;
103 }
104

```

Creature.hpp

```

5 #pragma once
6
7 #include <fstream>
8 #include <string>
9 #include "ground.h"
10
11 // class of abstract creatures
12 class Creature{
13 protected:
14     std::string _name;
15     int _power;
16     Creature (const std::string &str, int e = 0) :_name(str), _power(e) {}
17 public:
18     std::string name() const { return _name; }
19     bool alive() const { return _power > 0; }
20     void changePower(int e) { _power += e; }
21     virtual void transmute(Ground* &court) = 0;
22     virtual ~Creature () {}
23     static Creature* create(char ch, const std::string name, int p);
24 };
25
26 // class of green finches
27 class Greenfinch : public Creature {
28 public:
29     Greenfinch(const std::string &str, int e = 0) : Creature(str, e){}
30     void transmute(Ground* &court) override {
31         court = court->change(this);
32     }
33 };

```

```

35 // class of dune beetles
36 class DuneBeetle : public Creature {
37 public:
38     DuneBeetle(const std::string &str, int e = 0) : Creature(str, e){}
39     void transmute(Ground* &court) override {
40         court = court->change(this);
41     }
42 };
43
44 // class of squelchies
45 class Squelchy : public Creature {
46 public:
47     Squelchy(const std::string &str, int e = 0) : Creature(str, e){}
48     void transmute(Ground* &court) override{
49         court = court->change(this);
50     }
51 };
52

```

Creature.cpp

```

5 #include "creature.h"
6
7 Creature* Creature::create(char ch, const std::string name, int p)
8 {
9     switch (ch)
10     {
11         case 'G': return new Greenfinch(name, p);
12         case 'D': return new DuneBeetle(name, p);
13         case 'S': return new Squelchy(name, p);
14     }
15     return nullptr;
16 }
17

```

beadandó.cpp

```

int createPlanet(const std::string fileName, std::vector<Plant*> &plants){
    ifstream f(fileName);
    if(f.fail()){
        std::cout << "Sowwy but I can't find the file o.o" << std::endl;
        exit(1);
    }

    int plantDB, dayDB, nut;
    std::string name;
    char type;

    f >> plantDB;
    plants.resize(plantDB);

    for(int i=0; i<plantDB; ++i){
        f >> name >> type >> nut;
        plants[i] = Plant::create(name, type, nut);
    }

    f >> dayDB;
    return dayDB;
}

void destroyRad(Radiant* &rad){
    if(rad->isAlpha()){
        Alpha::destroy();
    } else if(rad->isDelta()){
        Delta::destroy();
    } else if (rad->isNoRad()){
        NoRad::destroy();
    }
}

```