Sec 001

```
1   #ifndef LIVESTOCK_H
2   #define LIVESTOCK_H
3
4   class Livestock {
5       protected :
6           float forage_amount;
7           int time_grazing;
8
9       public:
10          Livestock(); // constructor
11
12          // accessors / mutators
13
14          virtual float graze();
15
16          virtual void print_self()=0;
17          virtual Livestock* clone() const = 0;
18  };
19
20  #endif
21
```

virtual float graze(); → each child has its own overridden version

virtual void print_self()=0; → each child has its own

virtual Livestock* clone() const = 0; → Pure virtual function not defined in the parent → rely on children for the definition

Because we have one pure virtual this class is now an Abstract class → can't have an instance of it

```
1   #include "livestock.h"
2   #include <iostream>
3
4   using namespace std;
5
6   Livestock::Livestock() {
7       cout << "LIVESTOCK CONSTRUCTOR" << endl;
8       forage_amount = 1.25;
9       time_grazing = 5;
10  }
11
12  void Livestock::set_forage_amount(float a) { forage_amount = a; }
13  void Livestock::set_time_grazing(int t) { forage_amount = t; }
14  float Livestock::get_forage_amount() { return forage_amount; }
15  int Livestock::get_time_grazing() { return time_grazing; }
16
17
18  float Livestock::graze() {
19      cout << "LIVESTOCK GRAZE" << endl;
20      return time_grazing * forage_amount;
21  }
22
```

```
1   #ifndef COW_H
2   #define COW_H
3   #include "livestock.h"
4
5   class Cow : public Livestock{
6      private :
7           bool is_beef;
8           bool is_dairy;
9       public:
10
11          //accessor/mutators
12          cow(); //constructor
13
14      virtual float graze();
15
16      virtual Cow *  clone() const;
17      virtual void print_self();
18  };
19
20
21  #endif
22
```

→ cow will inherit from Livestock

→ No children so not being inherited

→ overriding function

→ not pure virtual

```cpp
#include "cow.h"
#include <iostream>

using namespace std;

Cow::Cow():Livestock() {
    cout << "COW CONSTRUCTOR" << endl;
    is_beef = true;
    is_dairy = false;
    time_grazing = 8;
}

void Cow::set_is_beef(bool b) {
    if(b) {
        is_beef = true;
        is_dairy = false;
    }
    else {
        is_beef = false;
        is_dairy = false;
    }
}

void Cow::set_is_dairy(bool d) {
    if(d) {
        is_dairy = true;
        is_beef = false;
    }
    else {
        is_dairy = false;
        is_beef = false;
    }
}

bool Cow::get_is_beef(){ return is_beef; }
bool Cow::get_is_dairy() { return is_dairy; }

float Cow::graze() {
    cout << "COW GRAZE" << endl;
    if (is_beef) {
        return (time_grazing*forage_amount)+5;
    }
    else if (is_dairy) {
        return (time_grazing*forage_amount)+7;
    }
    else {
        return time_grazing*forage_amount;
    }
}

void Cow::print_self() {
    if(is_beef)
        cout << "Beef Cow" << endl;
    else if(is_dairy)
        cout << "Dairy Cow" << endl;
    else
        cout << "Cow" << endl;
}

Cow* Cow::clone() const {
    return new Cow(*this);
}
```

reference to the current object we are in

copy constructor

↳ default version provided by the compiler shallow copy

same cow values but at new address

allocates memory from the heap

```
1   #ifndef HORSE_H
2   #define HORSE_H
3   #include "livestock.h"
4
5   class   Horse : public Livestock{
6      private :
7           int performance_rating;
8       public:
9            Horse();
10           // accessor/mutators
11           virtual float graze();
12           virtual Horse* clone() const;
13           virtual void print_self();
14  };
15
16  #endif
17
18
```

```cpp
#include "sheep.h"
#include <iostream>

using namespace std;

Sheep::Sheep():Livestock() {
    cout << "SHEEP CONSTRUCTOR" << endl;
    forage_amount = 1;
    time_grazing = 7;
}

void Sheep::print_self() {
    cout << "Sheep" << endl;
}

Sheep* Sheep::clone() const {
    return new Sheep(*this);
}
```

```
1   #ifndef PADDOCK_H
2   #define PADDOCK_H
3   #include "livestock.h"
4
5   class Paddock {
6       private:
7           float grass_amount;
8           int num_livestock;
9           Livestock** stock;
10          bool rotate;
11      public:
12          Paddock();
13          void set_grass_amount(float);
14          void set_rotate(bool);
15          void set_num_livestock(int);
16          void set_random_stock();
17          float get_grass_amount();
18          bool get_rotate();
19          void adjust_grass_amount_grazing();
20          void adjust_rotate();
21          void print_paddock();
22          Paddock(const Paddock &);
23          Paddock& operator=(const Paddock &);
24          ~Paddock();
25  };
26
27
28  #endif
29
```

→ array of livestocks that can polymorph into the children

```cpp
#include "paddock.h"
#include "horse.h"
#include "cow.h"
#include "sheep.h"
#include <iostream>
#include <cstdlib>
using namespace std;

Paddock::Paddock() {
    grass_amount = 10000;
    num_livestock = 0;
    stock = NULL;
    rotate = false;
}
void Paddock::set_grass_amount(float g) { grass_amount = g; }
void Paddock::set_rotate(bool r) { rotate = r; }
void Paddock::set_num_livestock(int num) { num_livestock = num; }

void Paddock::set_random_stock() {
    stock = new Livestock*[num_livestock];
    int choice = 0;
    for(int i=0; i<num_livestock; i++) {
        choice = rand() % 3;
        if(choice == 0)
            stock[i] = new Horse;
        else if (choice == 1)
            stock[i] = new Cow;
        else
            stock[i] = new Sheep;
    }
}

float Paddock::get_grass_amount() { return grass_amount; }
bool Paddock::get_rotate() { return rotate; }

void Paddock::adjust_grass_amount_grazing() {
    for(int i=0; i<num_livestock; i++) {
        grass_amount -= stock[i]->graze();
    }
}

void Paddock::adjust_rotate() {
    float amount_needed = 0;
    for(int i=0; i<num_livestock; i++) {
        amount_needed += stock[i]->graze();
    }
    if (grass_amount < (amount_needed*14)) {
        rotate = true;
    }
    else {
        rotate = false;
    }
}

void Paddock::print_paddock() {
    for(int i=0; i<num_livestock; i++) {
        stock[i]->print_self();
    }
}

Paddock::Paddock(const Paddock & copy) {
    grass_amount = copy.grass_amount;
    num_livestock = copy.num_livestock;
    stock = new Livestock*[num_livestock];
    for(int i=0; i<num_livestock; i++) {
        stock[i] = copy.stock[i]->clone();
    }
}
```

stock is an array of pointer so need dereference the object before calling the function

↳ at compile time we don't know the type pointed too so we need to poly morph the clone function to get the appropriate type

```
70  Paddock& Paddock::operator=(const Paddock & copy) {
71      grass_amount = copy.grass_amount;
72      if(stock != NULL) {
73          for(int i=0; i<num_livestock; i++) {
74              delete stock[i];
75          }
76          delete [] stock;
77          stock = NULL;
78          cout << stock << endl;
79      }
80      num_livestock = copy.num_livestock;
81      stock = new Livestock*[num_livestock];
82      for(int i=0; i<num_livestock; i++) {
83          stock[i] = copy.stock[i]->clone();
84      }
85  }
86
87  Paddock::~Paddock() {
88      for(int i=0; i<num_livestock; i++) {
89          delete stock[i];
90      }
91      delete [] stock;
92      stock = NULL;
93  }
```

```
1   #ifndef PASTURE_MANAGEMENT_H
2   #define PASTURE_MANAGMENT_H
3   #include "paddock.h"
4
5   class Pasture_Management {
6       private:
7           int num_paddocks;
8           Paddock* pasture;
9       public:
10          Pasture_Management();
11          void set_pasture(int);
12          Paddock get_paddock(int);
13          void day();
14          void suggest_rotation();
15          void print_pastures();
16          Pasture_Management(const Pasture_Management &);
17          Pasture_Management& operator=(const Pasture_Management &);
18          ~Pasture_Management();
19  };
20
21
22  #endif
23
```

```cpp
#include "pasture_management.h"
#include <iostream>
#include<cstdlib>
using namespace std;

Pasture_Management::Pasture_Management() {
    num_paddocks = 0;
    pasture = NULL;
}

void Pasture_Management::set_pasture(int num_paddocks) {
    this->num_paddocks = num_paddocks;
    pasture = new Paddock[this->num_paddocks];
    int num = 0;
    for(int i=0; i<num_paddocks; i++) {
        num = rand()%5;
        pasture[i].set_num_livestock(num);
        pasture[i].set_random_stock();
    }
}

Paddock Pasture_Management::get_paddock(int index) { return pasture[index]; }

void Pasture_Management::day() {
    for(int i=0; i<num_paddocks; i++) {
        pasture[i].adjust_grass_amount_grazing();
        pasture[i].adjust_rotate();
    }
}

void Pasture_Management::suggest_rotation() {
    for(int i=0; i<num_paddocks; i++) {
        if(pasture[i].get_rotate()) {
            cout << "Paddock " << i << " should be rotated." << endl;
        }
    }
}
void Pasture_Management::print_pastures() {
    for(int i=0; i<num_paddocks; i++) {
        cout << "PASTURE " << i << " GRASS LEVEL: " << pasture[i].get_grass_amount() <<
        endl;
        pasture[i].print_paddock();
        cout << endl;
    }
}
Pasture_Management::Pasture_Management(const Pasture_Management & copy) {
    num_paddocks = copy.num_paddocks;
    pasture = new Paddock[num_paddocks];
    for(int i=0; i<num_paddocks; i++) {
        pasture[i] = copy.pasture[i];
    }
}
Pasture_Management& Pasture_Management::operator=(const Pasture_Management & copy) {
    num_paddocks = copy.num_paddocks;
    if(pasture != NULL) {
        delete [] pasture;
    }
    pasture = new Paddock[num_paddocks];
    for(int i=0; i<num_paddocks; i++) {
        pasture[i] = copy.pasture[i];
    }
}
Pasture_Management::~Pasture_Management() {
    delete [] pasture;
}
```

```cpp
#include <iostream>
#include "pasture_management.h"

using namespace std;

int main() {

    Pasture_Management p;
    p.set_pasture(5);
    p.print_pastures();
    for(int i=0; i<7; i++){
        cout << "******** DAY " << i << " ********" << endl;
        p.day();
        p.print_pastures();
        p.suggest_rotation();
        cout << endl;
    }

    return 0;
}

```