CIS 330
C/C++ and Unix

Lecture 9

Access Control

Previously

We discussed concepts unique to C++ at a high level

We will talk about how classes are created and used in more detail

Debugging Example

Class

```
1. class <class name> {
2.     access_type_1:
3.         Member_1
4.     access_type_2:
5.         member_2
6.         member_3
7. } <object_name>
```

Example

```
1.  class Rectangle {
2.      private:
3.          int width;
4.          int height;
5.      public:
6.          void init_dims(int h, int w);
7.          int area();
8.  };
```

```
Rectangle square;
```

square is an **object** (or variable) of Rectangle **class** (data type)

Class

- Members are accessed using `.` – similar to how struct members are accessed

```
1. class Rectangle square; (or more commonly Rectangle square;)
2. square.init_dims(10, 10);
3. cout << "Area of square is " << square.area() << endl;
```

- where the public functions are defined as…

```
1. void Rectangle::init_dims(int w, int h)
2. { width = w; height = h; }
3. int Rectangle::area()
4. { return width * height; }
```

Because w and h are **private,** they can't be changed directly e.g., `square.w = w;`

- **::** is used to specify which class the members belong to (also known as the scope operator)

Hiding the Implementation

- Why?
  - Some functions are part of the interface the user needs to solve the problem related to the object – clear boundary between what is important and what is not. e.g., tools that are necessary for internal machination of the class should not be available for any user
  - Library designer can change the internal workings of the structure without worrying about how it will affect the user
  - Security, etc.

Access Control

- Three access types
  - public – all members declarations that follow are available to everyone to access
  - private – No one can access those members except you (the creator of the type)
  - protected – Behaves like private, except for inherited Classes (we will discuss this one in a bit)
- structs are (almost) identical to Classes
  - Things are public by default in structs
  - Things are private by default in Classes

Example

```
1.  struct Rectangle {
2.  private:
3.      float width;
4.      float height;
5.  public:
6.      float area() { return width * height; }
7.  };
8.  Rectangle A;
9.  A.width = 1.0;
10. A.height = 2.0;
11. cout << A.area() << endl;
```

Error

```
lecture010.cc:17:7: error: ‘float Rectangle::width’ is private within this context
     A.width = 1.0;
       ^~~~~
lecture010.cc:7:11: note: declared private here
     float width;
           ^~~~~
lecture010.cc:18:7: error: ‘float Rectangle::height’ is private within this context
     A.height = 2.0;
       ^~~~~~
lecture010.cc:8:11: note: declared private here
     float height;
           ^~~~~~
```

Private

- You can not directly access the member elements/functions if they are private
- How do you initialize an object (other than providing some public functions?
  - Constructors (we'll talk about this in a second)
  - What values do member functions get when created?
  - Undefined behavior (depends on the compiler/specification)

Example

```
1.  class Rectangle {
2.      private:
3.          int width;
4.          int height;
5.      public:
6.          void init_dims(int h, int w);
7.          int area();
8.  };
```

This is one way of providing initialization, but classes are like data types, so is there a way to initialize it "like a variable?"

Constructors

- In order to avoid uninitialized values within a class, we can specify a **constructor** to **automatically** initialize variables when an object is first created
- Has a pre-defined name – same as the class name and without any return type (not even void)
- Constructors **can not** be called explicitly (with "exceptions"), and are only executed once per object

Constructor Overloading

- Function overloading for constructors
- Compiler will automatically call the constructor with the matching parameters

Using a Constructor

Struct functions identically to class in this case because you've specified access for all members explicitly

```
1.   struct Rectangle {
2.   private:
3.       float width;
4.       float height;
5.   public:
6.       Rectangle(int w, int h) { width = w; height = h; }
7.       float area() { return width * height; }
8.   };
9.
10.  Rectangle A(10.0, 20.0);
11.  cout << A.area() << endl;
```

```
jeec@ix-dev: ~ 40$ ./a.out
200
```

Constructors

- You can have a class with a constructor and it still "works" - if you don't have one, the compiler will create one for you (the default constructor without any arguments)
- While you can have Classes without a constructor, **once you define one, you have to use it as it was defined**
- It's good to always provide a `**default'** constructor (a constructor with **NO** arguments)

- Valid
  - `Rectangle rectA;`
- Invalid
  - `Rectangle rectB();`
  - This will be seen as a function rectB (with no parameters) which returns a Rectangle as output, and not a constructor

Using a Constructor

```
1.  struct Rectangle {
2.  private:
3.      float width;
4.      float height;
5.  public:
6.      Rectangle(int w, int h) { width = w; height = h; }
7.      float area() { return width * height; }
8.  };
9.  Rectangle A;
10. cout << A.area() << endl;
```

Compile Error

```
lecture010.cc: In function 'int main()':
lecture010.cc:17:15: error: no matching function for call to 'Rectangle::Rectangle()'
     Rectangle A;
```

Constructors

```
1.   struct Rectangle {
2.   private:
3.       float width;
4.       float height;
5.   public:
6.       Rectangle() { width = 0.0; height = 0.0; }
7.       Rectangle(int w, int h) { width = w; height = h; }
8.       float area() { return width * height; }
9.   };
10. Rectangle A;
11. cout << A.area() << endl;
```

Will this now work?

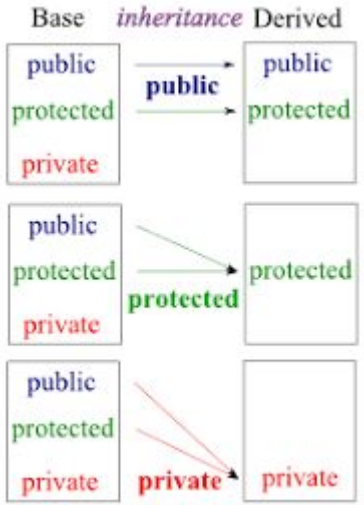
Constructors

```
1.   struct Rectangle {
2.   private:
3.       float width;
4.       float height;
5.   public:
6.       Rectangle() { width = 0.0; height = 0.0; }
7.       Rectangle(int w, int h) { width = w; height = h; }
8.       float area() { return width * height; }
9.   };
10. Rectangle A;
11. cout << A.area() << endl;
```

Yes - you've provided a default constructor and this will print 0

Questions?

Inheritance



Inheritance and Constructors

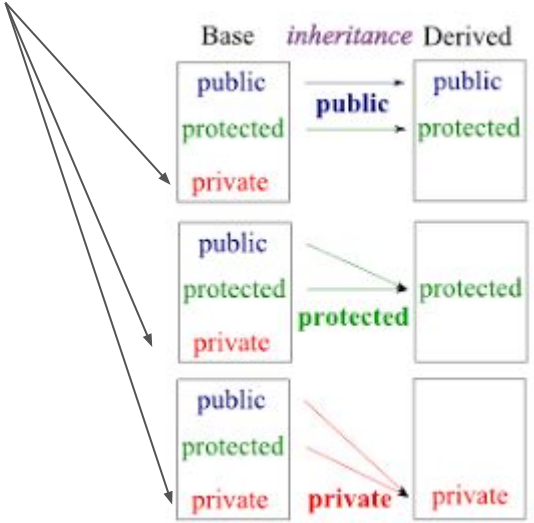
```
1.  struct Rectangle {
2.  private:
3.      float width;
4.      float height;
5.  protected:
6.      float some_random_value;
7.  public:
8.      Rectangle() { width = 0.0; height = 0.0; }
9.      Rectangle(int w, int h) { width = w; height = h; }
10.     float area() { return width * height; }
11. };
12.
13. struct Square : public Rectangle {
14.     Square() { width = 1.0; height = 1.0; }
15. };
16. Square B;
```

Compiler Error

```
lecture010.cc: In constructor ‘Square::Square()’:
lecture010.cc:19:16: error: ‘float Rectangle::width’
is private within this context
     Square() { width = 1.0; height = 1.0; }
                ^~~~~
```

Inheritance

What happens to this?



You still "have" them but they are not available to "access."

Inheritance

```
1.  struct Rectangle {
2.  private:
3.      float width;
4.      float height;
5.  protected:
6.      float some_random_value;
7.  public:
8.      Rectangle() { width = 0.0; height = 0.0; }
9.      Rectangle(int w, int h) { width = w; height = h; }
10.     float area() { return width * height; }
11. };
12. struct Square : public Rectangle {
13.     Square() { some_random_value = 3.14; }
14. };
15. Square B;
16. Cout << B.some_random_value << endl;
```

Compiler Error

```
lecture010.cc: In function 'int main()':
lecture010.cc:27:15: error: 'float Rectangle::some_random_value' is protected within this context
     cout << B.some_random_value << endl;
                ^~~~~~~~~~~~~~~~~
```

Remember, **protected** behaves like **private** for a given class

Inheritance

```
1.  struct Square : public Rectangle {
2.      Square() { some_random_value = 3.14; }
3.      Public:
4.          float get_random_value() {
5.              return some_random_value;
6.          }
7.  };
8.  Square B;
9.  Cout << B.get_random_value() << endl;
```

Questions?

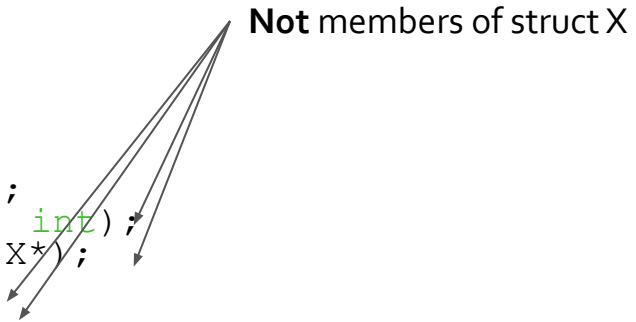
Friendship

- What if you want to grant access (to members in your structure) to a function that is NOT a member of your structure?
  - Declare that function as a **friend**
- Declare that function (you want to give access to) inside your class/structure as a friend
  - You can declare a (global) function, a function from another class/structure, or an entire class/structure as a friend

Friendship Example

Will this compile?

```
1.   struct Y {
2.       void f(X*);
3.   };
4.
5.   struct X {
6.   private:
7.       int i;
8.   public:
9.       void initialize();
10.      friend void g(X*, int);
11.      friend void Y::f(X*);
12.      friend struct Z;
13.      friend void h();
14.  };
```



Compile Error

```
lecture09friendship.cc:7:12: error: ‘X’ has not
been declared
     void f(X*);
            ^
```

Friendship Example

```
1.  struct X;

2.  struct Y {
3.      void f(X*);
4.  };

5.  struct X {
6.  private:
7.      int i;
8.  public:
9.      void initialize();
10.     friend void g(X*, int);
11.     friend void Y::f(X*);
12.     friend struct Z;
13.     friend void h();
14. };
```

Required, so struct Y knows what X is

friend can be used to simultaneously declare a function **and** give it friend status

Example

```
1.  struct X;
2.  struct Y {
3.      void f(X*);
4.  };
5.  void Y::f(X* x) {
6.      x->i = 47;
7.  }
8.  struct X {
9.  private:
10.     int i;
11. public:
12.     void initialize();
13.     friend void g(X*, int);
14.     friend void Y::f(X*);
15.     friend struct Z;
16.     friend void h();
17. };
```

Will this work?

Example

```
1.  struct X;
2.  struct Y {
3.      void f(X*);
4.  };
5.  void Y::f(X* x) {
6.      x->i = 47;
7.  }
8.  struct X {
9.  private:
10.     int i;
11. public:
12.     void initialize();
13.     friend void g(X*, int);
14.     friend void Y::f(X*);
15.     friend struct Z;
16.     friend void h();
17. };
```

No. X is used before its definition

Example

```
1. struct X;
2. struct Y {
3.     void f(X*);
4. };
5. struct X {
6. private:
7.     int i;
8. public:
9.     void initialize();
10.    friend void g(X*, int);
11.    friend void Y::f(X*);
12.    friend struct Z;
13.    friend void h();
14. };
15. void Y::f(X* x) {
16.    x->i = 47;
17. }
```

Example

```
1.  struct X;
2.  struct Y {
3.      void f(X*);
4.  };
5.  struct X {
6.  private:
7.      int i;
8.  public:
9.      void initialize();
10.     friend void g(X*, int);
11.     friend void Y::f(X*);
12.     friend struct Z;
13.     friend void h();
14. };
15. void Y::f(X* x) {
16.     x->i = 47;
17. }
```

Will this work?

```
18.    void X::initialize() {
19.        i = 0;
20.    }
   In main...
 1. X x;
 2. Y y;
 3. x.initialize();
 4. y.f(&x);
```

Example

```
1.  struct X;
2.  struct Y {
3.      void f(X*);
4.  };
5.  struct X {
6.  private:
7.      int i;
8.  public:
9.      void initialize();
10.     friend void g(X*, int);
11.     friend void Y::f(X*);
12.     friend struct Z;
13.     friend void h();
14. };
15. void Y::f(X* x) {
16.     x->i = 47;
17. }
```

Yes. x.i is 47 now.
This was only possible because Y::f(X*) was a friend of X

```
18.    void X::initialize() {
19.        i = 0;
20.    }
```

```
1.  X x;
2.  Y y;
3.  x.initialize();
4.  y.f(&x);
```

Example

```
1.  struct X;
2.  struct Y {
3.      void f(X*);
4.      void ff(X*);
5.  };
6.  struct X {
7.  private:
8.      int i;
9.  public:
10.     void initialize();
11.     friend void g(X*, int);
12.     friend void Y::f(X*);
13.     friend struct Z;
14.     friend void h();
15. };
16. void Y::ff(X* x) {
17.     x->i = 47;
18. }
```

Will this work?

```
18.    void X::initialize() {
19.        i = 0;
20.    }
```

```
1.  X x;
2.  Y y;
3.  x.initialize();
4.  y.f(&x);
```

Compile Error

```
lecture010.cc: In member function 'void Y::ff(X*)':
lecture010.cc:52:8: error: 'int X::i' is private
within this context
     x->i = 74;
        ^
```

Friendship

Global function

Entire struct

```
1.  struct X;
2.  struct Y {
3.      void f(X*);
4.  };
5.  struct X {
6.  private:
7.      int i;
8.  public:
9.      void initialize();
10.     friend void g(X*, int);
11.     friend void Y::f(X*);
12.     friend struct Z;
13.     friend void h();
14. };
15. void Y::f(X* x) {
16.     x->i = 47;
17. }
18. void g(X* x, int i) {
19.     x->i = i;
20. }
21. struct Z {
22. private:
23.     int j;
24. public:
25.     void initialize();
26.     void g(X* x);
27. };
28. void Z::initialize() {
29.     j = 99;
30. }
31. void Z::g(X* x) {
32.     x->i += j;
33. }
```

Example

```
1.   struct X {
2.   private: int i;
3.   public:
4.       void initialize();
5.       friend void g(X*, int);
6.       friend void Y::f(X*);
7.       friend struct Z;
8.       friend void h();
9.   };
18.  void X::initialize() {i = 0;}
1.   void Y::f(X* x) { x->i = 47; }
2.   struct Z { private:int j;
18.   public: void initialize();
19.           void g(X* x); };
20.  void Z::initialize() {j = 99;}
21.  void Z::g(X* x) {x->i += j;}
```

Will this work? What would happen?

```
11.  void h() {
12.      X x;
13.      x.i = 100;
14.  }
15.      X x;
16.      Y y;
17.      x.initialize();
18.      y.f(&x);
19.      Z z;
20.      z.initialize();
21.      z.g(&x);
22.      h();
```

Yes, it will work.

Example

```
1.   struct X {
2.   private: int i;
3.   public:
4.       void initialize();
5.       friend void g(X*, int);
6.       friend void Y::f(X*);
7.       friend struct Z;
8.       friend void h();
9.   };
18.  void X::initialize() {i = 0;}
1.   void Y::f(X* x) { x->i = 47; }
2.   struct Z { private:int j;
18.   public: void initialize();
19.           void g(X* x); };
20.  void Z::initialize() {j = 99;}
21.  void Z::g(X* x) {x->i += j;}
```

```
11. void h() {
12.     X x;
13.     x.i = 100;
14. }
15.     X x;
16.     Y y;
17.     x.initialize();
18.     y.f(&x); // x.i = 47
19.     Z z;
20.     z.initialize();
21.     z.g(&x); // x.i = 47 + 99 =146
22.     h(); // What about here?
```

Yes, it will work.

Example

```
1.  struct X {
2.  private: int i;
3.  public:
4.      void initialize();
5.      friend void g(X*, int);
6.      friend void Y::f(X*);
7.      friend struct Z;
8.      friend void h();
9.  };
18. void X::initialize() {i = 0;}
1.  void Y::f(X* x) { x->i = 47; }
2.  struct Z { private:int j;
18.  public: void initialize();
19.          void g(X* x); };
20. void Z::initialize() {j = 99;}
21. void Z::g(X* x) {x->i += j;}
```

```
11. void h() {
12.     X x;
13.     x.i = 100;
14. }
15.     X x;
16.     Y y;
17.     x.initialize();
18.     y.f(&x); // x.i = 47
19.     Z z;
20.     z.initialize();
21.     z.g(&x); // x.i = 47 + 99 =146
22.     h(); // Nothing else happens
```

Nested Friendship

- In the older version of C++ (C++98 and C++03), nested class cannot access private and protected members of enclosing class by default
- However, that was updated with the newer version of C++ - a nested class is a member, and as such has the same access rights as any other member.
- Chapter 5 - Nested friends need to be updated with this

Older C++ Versions

```
1.  struct Holder {
2.  private:
3.      int a[sz];
4.  public:
5.      void initialize();
6.      struct Pointer;
7.      friend struct Pointer;
8.      struct Pointer {
9.      private:
10.         Holder* h;
11.         int* p;
12.     public:
13.         void initialize(Holder* h);
14.         void next(); void previous();
15.         void top();  void end();
16.         int read(); void set(int i);
17.     };
```

Required before

Newer C++ Versions

```
1.  struct Holder {
2.  private:
3.      int a[sz];
4.  public:
5.      void initialize();
6.      // struct Pointer;
7.      // friend struct Pointer;
8.      struct Pointer {
9.      private:
10.         Holder* h;
11.         int* p;
12.     public:
13.         void initialize(Holder* h);
14.         void next(); void previous();
15.         void top();  void end();
16.         int read(); void set(int i);
17.     };
```

Works fine

Why Do We Need Friends?

- C++ is a HYBRID OO language, not a PURE one, so there may be cases when a function (that is not related to the object) needs access to a private member
- C++ is designed to be pragmatic, not to aspire to an abstract ideal

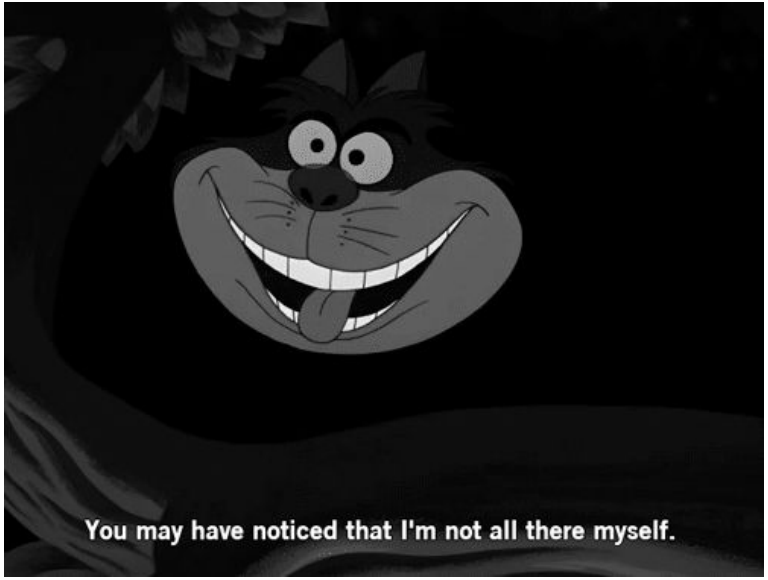
Questions?

Hiding the Implementation From Everyone (sort of)

- What if you don't want your user to see the implementation AT ALL?
- For example, what if your code is related to encryption – knowing parts of the Class might make it more vulnerable

Cheshire Cat

- Everything about the implementation disappears except for a single pointer – the "smile."



Header File

```
1. #ifndef HANDLE_H
2. #define HANDLE_H
3. class Handle {
4.     struct Cheshire;
5.     Cheshire *smile;
6. public:
7.     void initialize();
8.     void cleanup();
9.     int read();
0.     void change(int);
1. };
2. #endif
```

Header file can be seen by anyone – otherwise you won't know how to use it

The REAL private members are not visible

The real private members are "hidden" inside the Cheshire struct

Cheshire Cat

- Implement the Class in the source code
- Compile it
- Provide it as a library/API with the header to provide the interface (and include what the interface is used for)
- The user has no idea what's actually inside the Class (i.e., the private members)

Cheshire Cat

```
#include "Handle.h"
struct Handle::Cheshire {
  int i;
};
void Handle::initialize() {
  smile = new Cheshire;
  smile->i = 0;
}
void Handle::cleanup() {
  delete smile;
}
int Handle::read() {
  return smile->i;
}
void Handle::change(int x) {
  smile->i = x;
}
```

Initialization - Constructor

- Guaranteed initialization with the constructor(s) (provided you wrote them)
- As the Class creator you should ALWAYS have a default constructor (even if your code will compile without it)

Cleanup - Destructor

- Similar to the idea behind constructors, **destructors** exist to do the cleanup
- Destructor is called when the object goes out of scope (e.g., the function where it was declared finishes)
- **No arguments** for the destructor

Destructor Example

```
1.  class X {
2.      int i;
3.  public:
4.      X() { i = 1; }
5.
6.      // This is the destructor
7.      ~X() { cout << "The end " << endl; };
8.
9.      void print_X() { cout << i << endl; }
10. };
```

Questions?