CS 32 Week 1
Discussion 2E

Srinath

TA Introduction

- **Srinath**
- MS CS @ UCLA, Currently in my 2nd year.
- Email : srinath@g.ucla.edu, (I will try to reply ASAP)
- Hobbies : Badminton, Volleyball, Chess, Photography,...

LA Introduction

- **Jed Miguel**
- **Hairan Liang**

Class Introductions

Name
Major and Year
(Optionally) hobbies, fun facts etc.

Logistics

- **Class website :** http://web.cs.ucla.edu/classes/winter23/cs32/
- **Discussion slides :** Will be posted to bruinlearn.
- also at http://web.cs.ucla.edu/~srinath/teaching.html
- **TA office hours :** Thursdays (3:30 p.m - 6:30 p.m)
- Fridays (4:30 p.m - 5:30 p.m)
- **LA office hours :** Hairan **-** Wednesdays (4:30 p.m - 5:30 p.m)
  Jed - Tuesday (5:30 - 6:30 p.m)
  Thursday (2:30 - 3:30 p.m)

Friday Discussion’s

- Important announcements
- Lecture review
- Practice problems (a.k.a Worksheets)
- Anything else which you might need help with.

Announcements

- Project 1 is Up!
- Due : 11:00 PM, Wednesday, January 18

Outline

- Review Constructors
- Order of Construction
- Const member functions
- #include guard
- Circular Dependency

Struct : Constructor

- What is a constructor?

Struct : Constructor

- Every object defined using Struct/Class needs a constructor to create the object.
- Generally used to initialize the data members.

```
struct Employee {
        string name;
        double salary;
    private:
        int age;
};

int main() {
    Employee emp1, emp2;
}
```

Is this invalid then?

Struct : Constructor

- Every object defined using Struct/Class needs a constructor to create the object.
- Generally used to initialize the data members.

Note that constructor doesn't have a return type (a special function!!)

<StructName>() → specification of default constructor

```
struct Employee {
        string name;
        double salary;
    private:
        int age;

    public:
    Employee(){}
};

int main() {
    Employee emp1, emp2;
}
```

Is this invalid then?

- It's valid. Compiler defines a default Constructor for us

Struct : Constructor

Can we define our own constructor?

-

If we write our own constructor, will the compiler still write a default one for us?

-

Can we define constructor with parameters?

-

Can we define multiple constructors?

-

Does ‘Employee’ struct have a default constructor?

-

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    Employee(string name){
        m_name = name;
    }

    Employee(int age){
        m_age = age;
    }
};
```

Struct : Constructor

Can we define our own constructor?
- *Yes!*

If we write our own constructor, will the compiler still write a default one for us?
- *No!*

Can we define constructor with parameters?
- *Yes!*

Can we define multiple constructors?
- *Yes!*

Does ‘Employee’ struct have a default constructor?
- *No. Default is one is with no parameters*
  => Employee() {...}

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    Employee(string name){
        m_name = name;
    }

    Employee(int age){
        m_age = age;
    }
};
```

Struct : Constructor

```
Employee emp;          // Will this compile?
-
Employee emp("jack");  // Will this compile?
-
Employee emp(100);     // Will this compile?
-
Employee emp(200.0);   // Will this compile?
-
```

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    Employee(string name){
        m_name = name;
    }

    Employee(int age){
        m_age = age;
    }
};
```

Struct : Constructor

Employee emp; // Will this compile?
- No, it doesn't have a default constructor.

Employee emp("jack"); // Will this compile?
- Yes.

Employee emp(100); // Will this compile?
- Yes.

Employee emp(200.0); // Will this compile?
- No, it doesn't have constructor with double as a parameter.

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    Employee(string name){
        m_name = name;
    }

    Employee(int age){
        m_age = age;
    }
};
```

Struct : Constructor

```
Employee emp;          // Will this compile?
```

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    Employee(){
        m_salary = 0.0;
        m_age = 1;
    }

    Employee(string name){
        m_name = name;
    }

    Employee(int age){
        m_age = age;
    }
};
```

Struct : Constructor

```
Employee emp;          // Will this compile?
```

- Yes.

```
struct Employee {
        string m_name;
        double m_salary;
    private:
        int m_age;

    public:
    Employee(){
        m_salary = 0.0;
        m_age = 1;
    }

    Employee(string name){
        m_name = name;
    }

    Employee(int age){
        m_age = age;
    }
};
```

Order of Construction

1. ?

Order of Construction

1. —----------
2. ?

Order of Construction

1. —----------
2. Construct the data members, consulting the **member initialization list**
    a. If data member is not listed in member initialization list
        i. If built in type : ?
        ii. If class type : ?

Order of Construction

1. —----------
2. Construct the data members, consulting the **member initialization list**
    a. If data member is not listed in member initialization list
        i. If built in type : left uninitialized
        ii. If class type : **default constructor** is called
3. ?

Order of Construction

1. —----------
2. Construct the data members, consulting the **member initialization list**
    a. If data member is not listed in member initialization list
        i. If built in type : left uninitialized
        ii. If class type : **default constructor** is called
3. Execute the body of the constructor

Order of Construction

```
class Author {
        string m_name;
        int m_rating;
    public :
        Author(string name, int rating){
                m_name = name;
                m_rating = rating;
        }
};

class Book {
        string m_title;
        Author m_author;
     public :
        Book(string title, string author_name, int author_rating){
                m_title = title;
                m_author = Author(author_name, author_rating);
        }
};
```

Will this compile?

Order of Construction

```
class Author {
        string m_name;
        int m_rating;
    public :
        Author(string name, int rating){
                m_name = name;
                m_rating = rating;
        }
};

class Book {
        string m_title;
        Author m_author;
     public :
        Book(string title, string author_name, int author_rating){
                m_title = title;
                m_author = Author(author_name, author_rating);
        }
};
```

Will this compile?

- No, ‘Author’ doesn’t have a default constructor.

Order of Construction

```
class Author {
        string m_name;
        int m_rating;
    public :
        Author(string name, int rating){
                m_name = name;
                m_rating = rating;
        }
};

class Book {
        string m_title;
        Author m_author;
     public :
        Book(string title, string author_name, int author_rating) : m_author(author_name, author_rating) {
                m_title = title;
        }
};
```

Member initialization list fixes it.

Order of Construction

```
class Author {
        string m_name;
        int m_rating;
    public :
        Author(){
                m_name = “random peacock”;
                m_rating = 0;
        }
        Author(string name, int rating){
                m_name = name;
                m_rating = rating;
        }
};

class Book {
        string m_title;
        Author m_author;
     public :
        Book(string title, string author_name, int author_rating){
                m_title = title;
                m_author = Author(author_name, author_rating);
        }
};
```

Will this compile?

Order of Construction

```
class Author {
        string m_name;
        int m_rating;
    public :
        Author(){
                m_name = “random peacock”;
                m_rating = 0;
        }
        Author(string name, int rating){
                m_name = name;
                m_rating = rating;
        }
};

class Book {
        string m_title;
        Author m_author;
     public :
        Book(string title, string author_name, int author_rating){
                m_title = title;
                m_author = Author(author_name, author_rating);
        }
};
```

Will this compile?

- Yes, we do have the default constructor.

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary(){
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0}

    bool isOnStrike(Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
}
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp(“Einstein”);
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary(){
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0}

    bool isOnStrike(Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
}
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

-

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary(){
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0;}

    bool isOnStrike(const Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
};
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

- No, The compiler doesn't know if method getSalary() modifies Employee.

How can we fix this?

-

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary(){
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0;}

    bool isOnStrike(const Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
};
```

Struct : with Functions

We can also pass struct/class as arguments to member functions.

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

- No, The compiler doesn't know if method getSalary() modifies Employee.

How can we fix this?

- 'const' member functions

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary(){
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0;}

    bool isOnStrike(const Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
};
```

Struct : Const Member Functions

'Const' member functions are not supposed to modify the object.
Compiler knows that the object is not modified.

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(&emp) << endl;
}
```

What if we use const reference for Employee.
It means the function should not be able to change the Employee object pointed by.

Will this compile?

- No, The compiler doesn't know if method getSalary() modifies Employee.

How can we fix this?

- 'const' member functions

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary() const {
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0;}

    bool isOnStrike(const Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
};
```

Struct : Const Member Functions

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(emp) << endl;
}
```

Will this compile?

-

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary() const {
        m_salary++;
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0}

    bool isOnStrike(const Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
}
```

Struct : Const Member Functions

```
int main() {
        UC ucla;
        Employee emp("Einstein");
        cout<< ucla.isOnStrike(emp) << endl;
}
```

Will this compile?

- No, const member function shouldn't modify any variable.

```
struct Employee {
        string m_name;
    private:
        double m_salary;
        int m_age;

    public:
    Employee(string name){
        m_name = name;
        m_salary = 0.0;
        m_age = 1;
    }

    double getSalary() const {
        m_salary++;
        return m_salary;
    }
};

struct UC {
    double m_budget;
    public:
    UC(){m_budget = 9000.0}

    bool isOnStrike(const Employee * emp){
        double sal = emp->getSalary();
        return sal<100.0;
    }
}
```

#include guard

..... ?

#include guard

“Point.h”

```
class Point{
        int m_x;
        int m_y;
};
```

“Circle.h”

```
#include "Point.h"

class Circle{
        Point m_p;
        int m_r;
};
```

“Main.cpp”

```
#include "Point.h"
#include "Circle.h"

int main(){
        return 0;
}
```

Will this compile?

#include guard

“Point.h”

```
class Point{
        int m_x;
        int m_y;
};
```

“Circle.h”

```
#include "Point.h"

class Circle{
        Point m_p;
        int m_r;
};
```

“Main.cpp”

```
#include "Point.h"
#include "Circle.h"

int main(){
        return 0;
}
```

Will this compile? - No
What is the fix?

#include guard

“Point.h”

```
#ifndef POINT_INCLUDED
#define POINT_INCLUDED

class Point{
        int m_x;
        int m_y;
};

#endif
```

“Circle.h”

```
#include "Point.h"

class Circle{
        Point m_p;
        int m_r;
};
```

“Main.cpp”

```
#include "Point.h"
#include "Circle.h"

int main(){
        return 0;
}
```

Will this compile? - No
What is the fix? - Include Guard

#include guard

A construct to make sure each header file is included once for each source file.

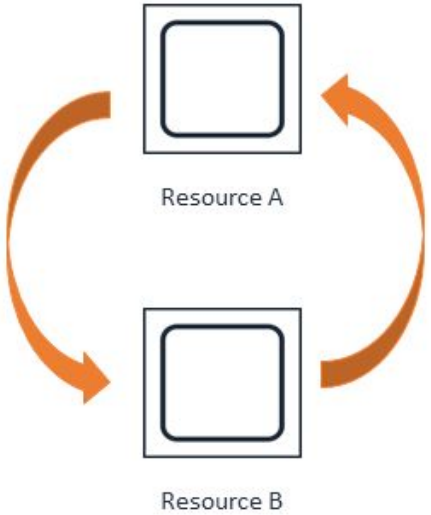
```
“XXX.h”
#ifndef XXX_INCLUDED
#define XXX_INCLUDED
class XXX{
   ….
};
…
#endif
```

Circular Dependency

..... ?

Circular Dependency

Resource A is dependent on Resource B



Resource B is dependent on Resource A

Circular Dependency

“Author.h”

```
#ifndef Author_INCLUDED
#define Author_INCLUDED

#include <string>
#include "Book.h"

class Author {
        std::string m_name;
        int m_rating;
        Book* books[10];
public:
        Author(std::string name, int rating);
};

#endif
```

“Book.h”

```
#ifndef Book_INCLUDED
#define Book_INCLUDED

#include <string>
#include "Author.h"

class Book {
        std::string m_title;
        Author m_author;
public:
        Book(std::string title, std::string author_name, int author_rating);
};

#endif
```

“Main.cpp”

```
#include "Author.h"
#include "Book.h"

int main(){
        Book b("cs32", "smallberg", 5);
        return 0;
}
```

Will this compile?

Circular Dependency

“Author.h”

```
#ifndef Author_INCLUDED
#define Author_INCLUDED

#include <string>
#include "Book.h"

class Author {
        std::string m_name;
        int m_rating;
        Book* books[10];
public:
        Author(std::string name, int rating);
};

#endif
```

“Book.h”

```
#ifndef Book_INCLUDED
#define Book_INCLUDED

#include <string>
#include "Author.h"

class Book {
        std::string m_title;
        Author m_author;
public:
        Book(std::string title, std::string author_name, int author_rating);
};

#endif
```

“Main.cpp”

```
#include "Author.h"
#include "Book.h"

int main(){
        Book b("cs32", "smallberg", 5);
        return 0;
}
```

Will this compile? - No, circular dependency!
How to fix this?

Circular Dependency

“Author.h”

```
#ifndef Author_INCLUDED
#define Author_INCLUDED

#include <string>
class Book;

class Author {
        std::string m_name;
        int m_rating;
        Book* books[10];
public:
        Author(std::string name, int rating);
};

#endif
```

“Book.h”

```
#ifndef Book_INCLUDED
#define Book_INCLUDED

#include <string>
#include "Author.h"

class Book {
        std::string m_title;
        Author m_author;
public:
        Book(std::string title, std::string author_name, int author_rating);
};

#endif
```

“Main.cpp”

```
#include "Author.h"
#include "Book.h"

int main(){
        Book b("cs32", "smallberg", 5);
        return 0;
}
```

Will this compile? - No, circular dependency!
How to fix this? - Forward declaration

Circular Dependency

“Author.h”

```
#ifndef Author_INCLUDED
#define Author_INCLUDED

#include <string>
class Book;

class Author {
        std::string m_name;
        int m_rating;
        Book* books[10];
public:
        Author(std::string name, int rating);
};

#endif
```

“Book.h”

```
#ifndef Book_INCLUDED
#define Book_INCLUDED

#include <string>
#include "Author.h"

class Book {
        std::string m_title;
        Author m_author;
public:
        Book(std::string title, std::string author_name, int author_rating);
};

#endif
```

“Main.cpp”

```
#include "Author.h"
#include "Book.h"

int main(){
        Book b("cs32", "smallberg", 5);
        return 0;
}
```

You might need this for project 1 :)

Will this compile? - No, circular dependency!
How to fix this? - Forward declaration

Summary

- Order of Construction
- Const member functions
- #include guard
- Circular Dependency

Questions?