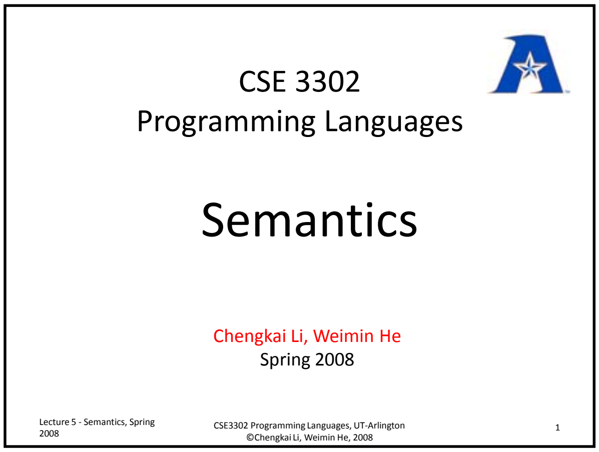
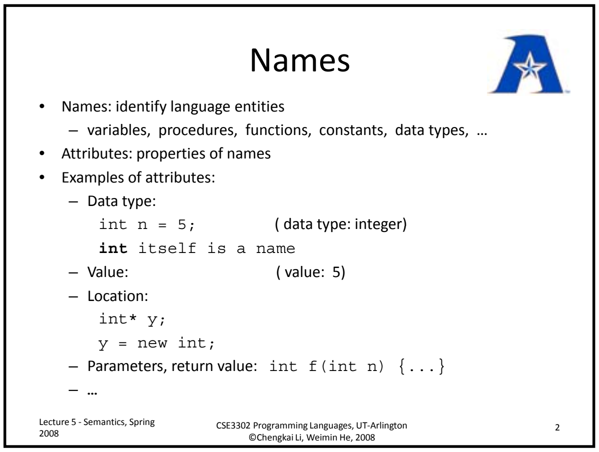


CSE 3302 Programming Languages

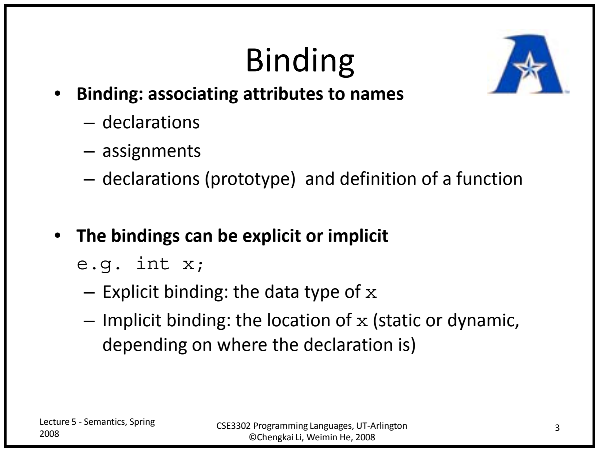
Semantics

Chengkai Li, Weimin He
Spring 2008



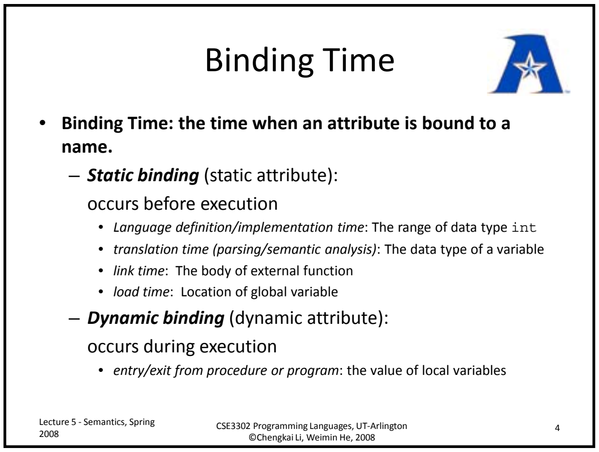
Names

- Names: identify language entities
  - variables, procedures, functions, constants, data types, …
- Attributes: properties of names
- Examples of attributes:
  - Data type:
    `int n = 5;` (data type: integer)
    `int` `itself is a name`
  - Value: (value: 5)
  - Location:
    ```
    int* y;
    y = new int;
    ```
  - Parameters, return value: `int f(int n) {...}`
  - …



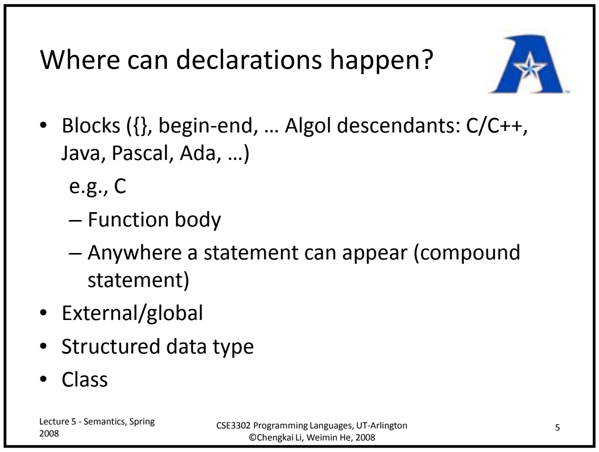
Binding

- **Binding: associating attributes to names**
  - declarations
  - assignments
  - declarations (prototype) and definition of a function
- **The bindings can be explicit or implicit**
  `e.g. int x;`
  - Explicit binding: the data type of `x`
  - Implicit binding: the location of `x` (static or dynamic, depending on where the declaration is)



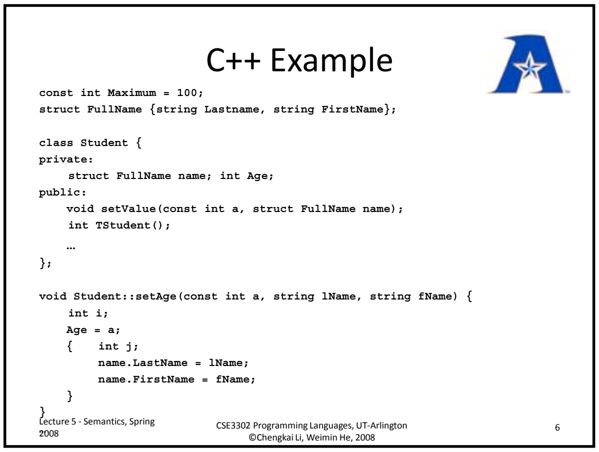
Binding Time

- **Binding Time: the time when an attribute is bound to a name.**
  - ***Static binding*** (static attribute):
    occurs before execution
    - *Language definition/implementation time*: The range of data type `int`
    - *translation time (parsing/semantic analysis)*: The data type of a variable
    - *link time*: The body of external function
    - *load time*: Location of global variable
  - ***Dynamic binding*** (dynamic attribute):
    occurs during execution
    - *entry/exit from procedure or program*: the value of local variables



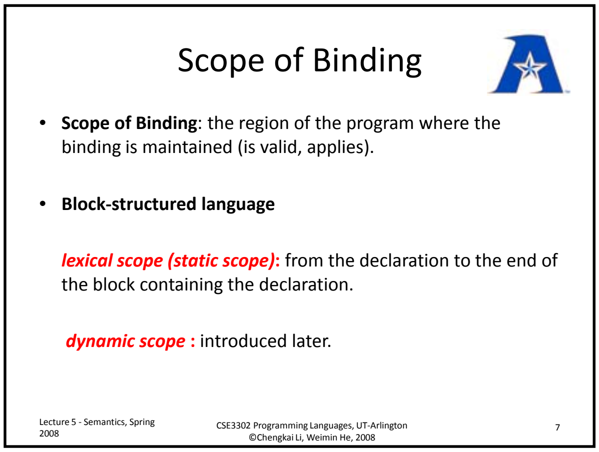
Where can declarations happen?

- Blocks ({}, begin-end, … Algol descendants: C/C++, Java, Pascal, Ada, …)
  e.g., C
  - Function body
  - Anywhere a statement can appear (compound statement)
- External/global
- Structured data type
- Class



C++ Example

```
const int Maximum = 100;
struct FullName {string Lastname, string FirstName};

class Student {
private:
    struct FullName name; int Age;
public:
    void setValue(const int a, struct FullName name);
    int TStudent();
    …
};

void Student::setAge(const int a, string lName, string fName) {
    int i;
    Age = a;
    {   int j;
        name.LastName = lName;
        name.FirstName = fName;
    }
}
```



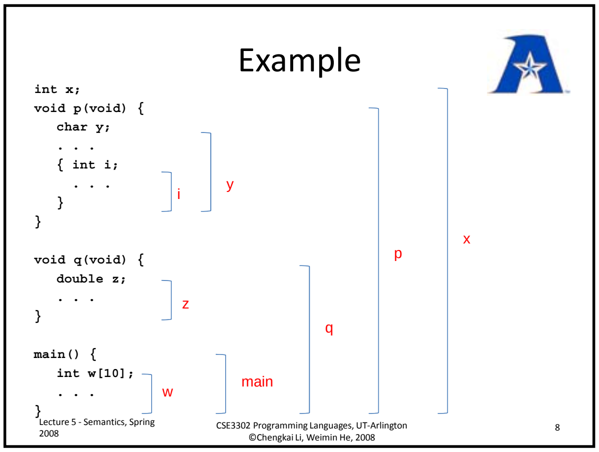
Scope of Binding

- **Scope of Binding**: the region of the program where the binding is maintained (is valid, applies).

- **Block-structured language**

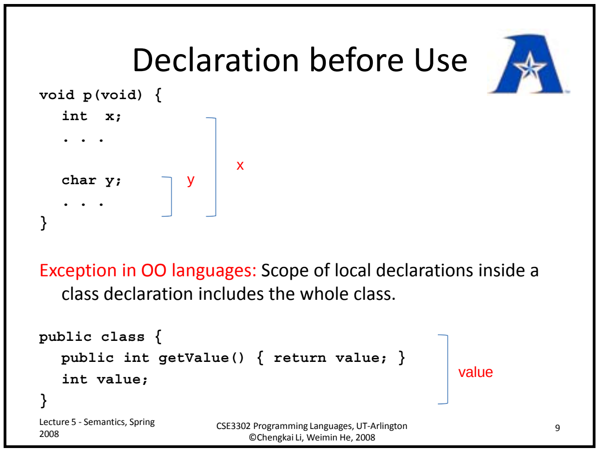
  lexical scope (static scope)**:** from the declaration to the end of the block containing the declaration.

  dynamic scope **:** introduced later.



Example

```
int x;
void p(void) {
   char y;
   . . .
   { int i;
     . . .
   }
}

void q(void) {
   double z;
   . . .
}

main() {
   int w[10];
   . . .
}
```

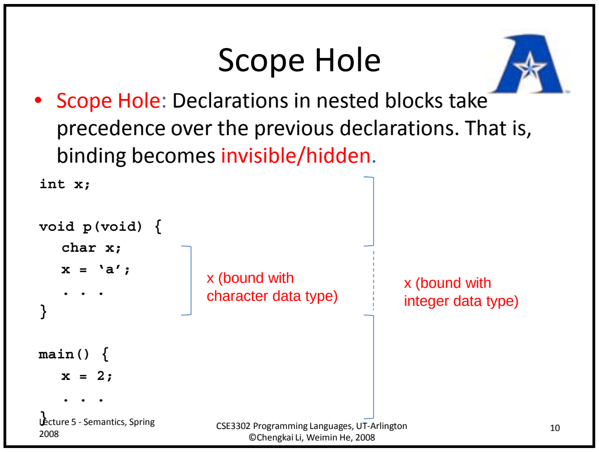


Declaration before Use

```
void p(void) {
   int  x;
   . . .

   char y;
   . . .
}
```

Exception in OO languages: Scope of local declarations inside a class declaration includes the whole class.

```
public class {
   public int getValue() { return value; }
   int value;
}
```

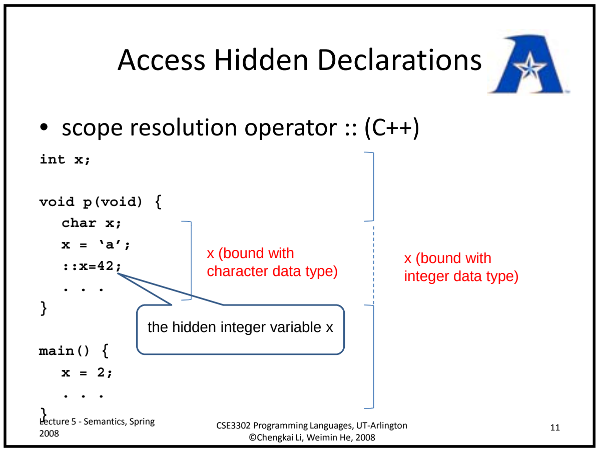


Scope Hole

- Scope Hole: Declarations in nested blocks take precedence over the previous declarations. That is, binding becomes invisible/hidden.

```
int x;

void p(void) {
   char x;
   x = 'a';
   . . .
}

main() {
   x = 2;
   . . .
}
```

x (bound with character data type)
x (bound with integer data type)

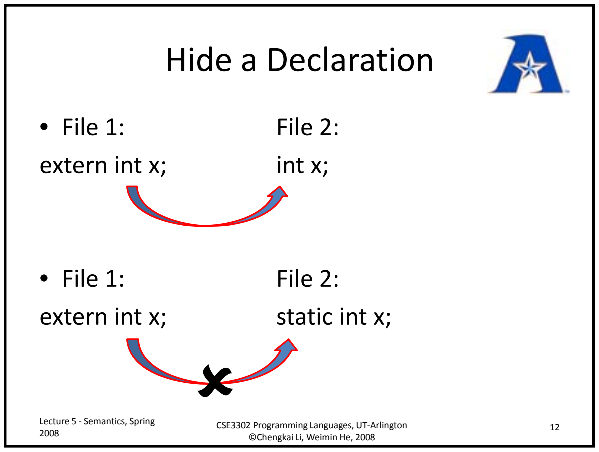


Access Hidden Declarations

- scope resolution operator :: (C++)

```
int x;

void p(void) {
   char x;
   x = 'a';
   ::x=42;
   . . .
}

main() {
   x = 2;
   . . .
}
```

x (bound with character data type)
x (bound with integer data type)
the hidden integer variable x



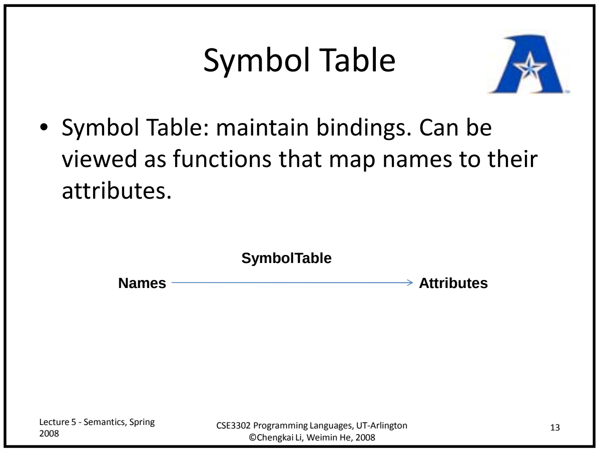
Hide a Declaration

- File 1: File 2:

extern int x; int x;

- File 1: File 2:

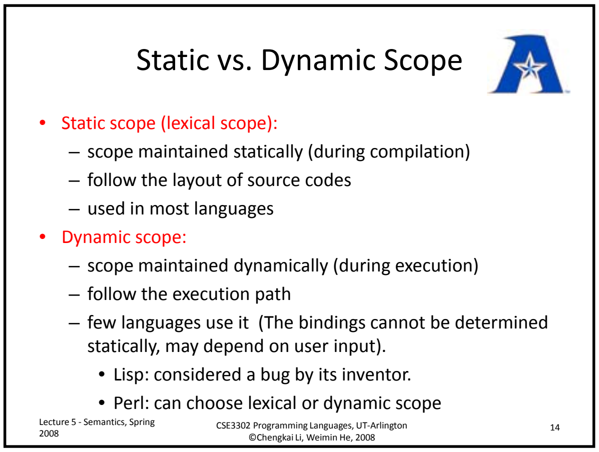
extern int x; static int x;



Symbol Table

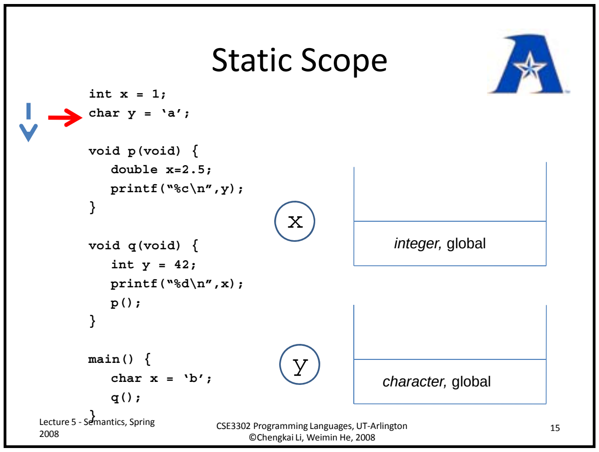
- Symbol Table: maintain bindings. Can be viewed as functions that map names to their attributes.

SymbolTable
Names
Attributes



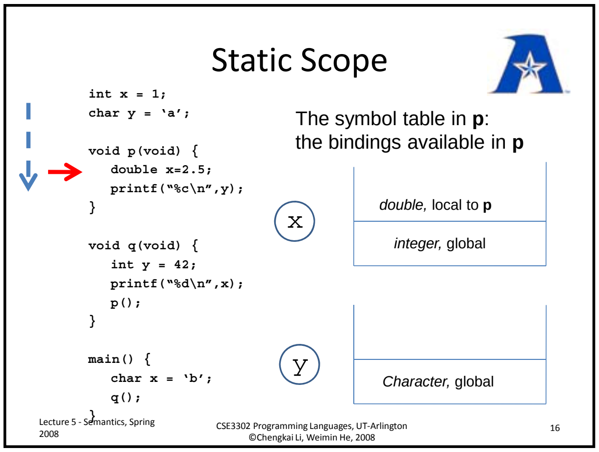
Static vs. Dynamic Scope

- Static scope (lexical scope):
  - scope maintained statically (during compilation)
  - follow the layout of source codes
  - used in most languages
- Dynamic scope:
  - scope maintained dynamically (during execution)
  - follow the execution path
  - few languages use it (The bindings cannot be determined statically, may depend on user input).
    - Lisp: considered a bug by its inventor.
    - Perl: can choose lexical or dynamic scope



Static Scope

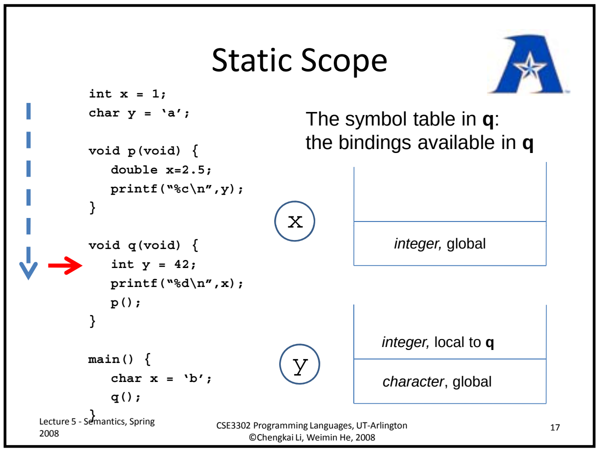
```
int x = 1;
char y = ‘a’;

void p(void) {
   double x=2.5;
   printf(“%c\n”,y);
}

void q(void) {
   int y = 42;
   printf(“%d\n”,x);
   p();
}

main() {
   char x = ‘b’;
   q();
}
```



Static Scope

```
int x = 1;
char y = ‘a’;

void p(void) {
   double x=2.5;
   printf(“%c\n”,y);
}

void q(void) {
   int y = 42;
   printf(“%d\n”,x);
   p();
}

main() {
   char x = ‘b’;
   q();
}
```

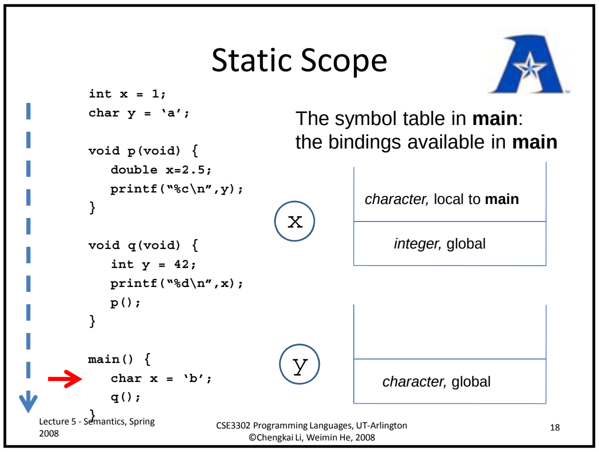
The symbol table in **p**:
the bindings available in **p**



Static Scope

```
int x = 1;
char y = ‘a’;

void p(void) {
   double x=2.5;
   printf(“%c\n”,y);
}

void q(void) {
   int y = 42;
   printf(“%d\n”,x);
   p();
}

main() {
   char x = ‘b’;
   q();
}
```

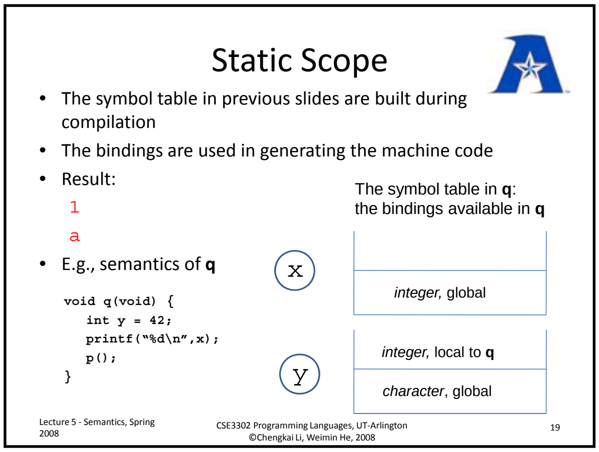
The symbol table in **q**:
the bindings available in **q**



Static Scope

```
int x = 1;
char y = ‘a’;

void p(void) {
   double x=2.5;
   printf(“%c\n”,y);
}

void q(void) {
   int y = 42;
   printf(“%d\n”,x);
   p();
}

main() {
   char x = ‘b’;
   q();
}
```

The symbol table in **main**:
the bindings available in **main**



Static Scope

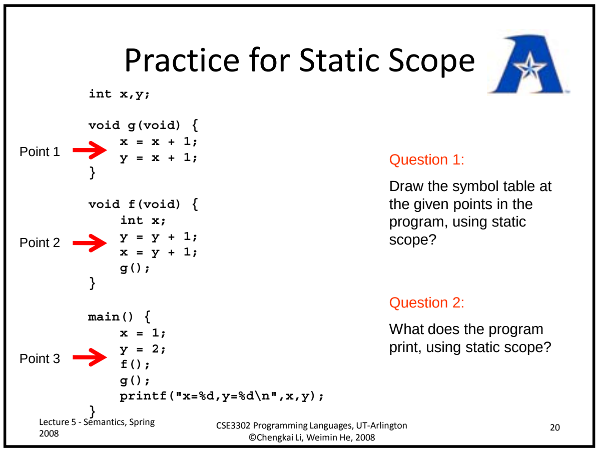
- The symbol table in previous slides are built during compilation
- The bindings are used in generating the machine code
- Result:

  1

  a
- E.g., semantics of **q**

```
void q(void) {
   int y = 42;
   printf("%d\n",x);
   p();
}
```

The symbol table in **q**:
the bindings available in **q**



Practice for Static Scope

```
int x,y;

void g(void) {
   x = x + 1;
   y = x + 1;
}

void f(void) {
   int x;
   y = y + 1;
   x = y + 1;
   g();
}

main() {
   x = 1;
   y = 2;
   f();
   g();
   printf("x=%d,y=%d\n",x,y);
}
```

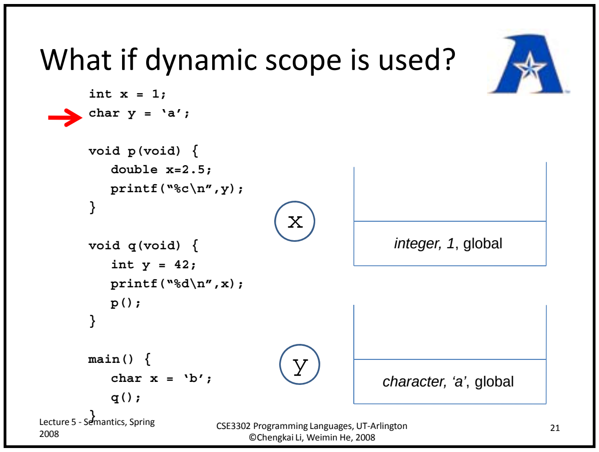
Point 1
Point 2
Point 3

Question 1:

Draw the symbol table at the given points in the program, using static scope?

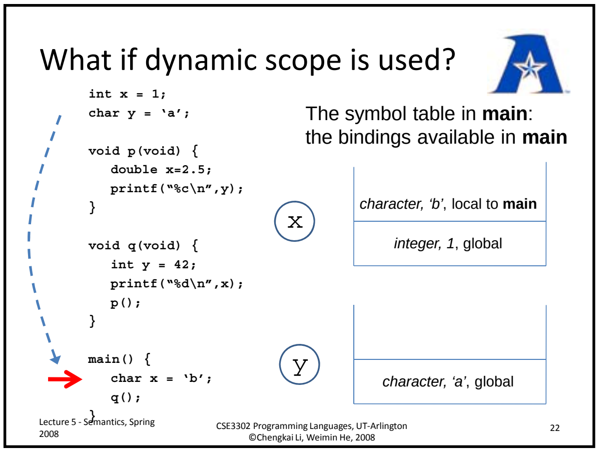
Question 2:

What does the program print, using static scope?



What if dynamic scope is used?

```
int x = 1;
char y = 'a';

void p(void) {
   double x=2.5;
   printf("%c\n",y);
}

void q(void) {
   int y = 42;
   printf("%d\n",x);
   p();
}

main() {
   char x = 'b';
   q();
}
```

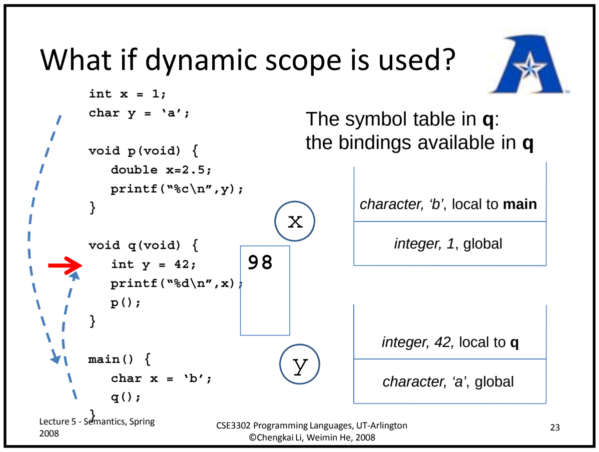


What if dynamic scope is used?

```
int x = 1;
char y = 'a';

void p(void) {
   double x=2.5;
   printf("%c\n",y);
}

void q(void) {
   int y = 42;
   printf("%d\n",x);
   p();
}

main() {
   char x = 'b';
   q();
}
```

The symbol table in **main**:
the bindings available in **main**

x
character, *'b'*, local to **main**
integer, *1*, global
y
character, *'a'*, global



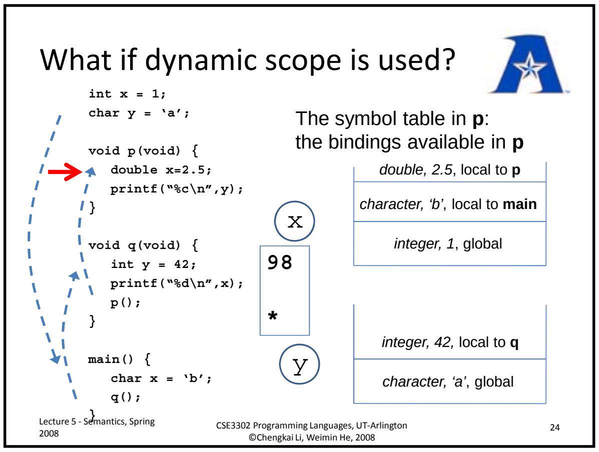
What if dynamic scope is used?

```
int x = 1;
char y = 'a';

void p(void) {
   double x=2.5;
   printf("%c\n",y);
}

void q(void) {
   int y = 42;
   printf("%d\n",x);
   p();
}

main() {
   char x = 'b';
   q();
}
```

The symbol table in **q**:
the bindings available in **q**

```
98
```

x
character, *'b'*, local to **main**
integer, *1*, global
y
integer, *42*, local to **q**
character, *'a'*, global



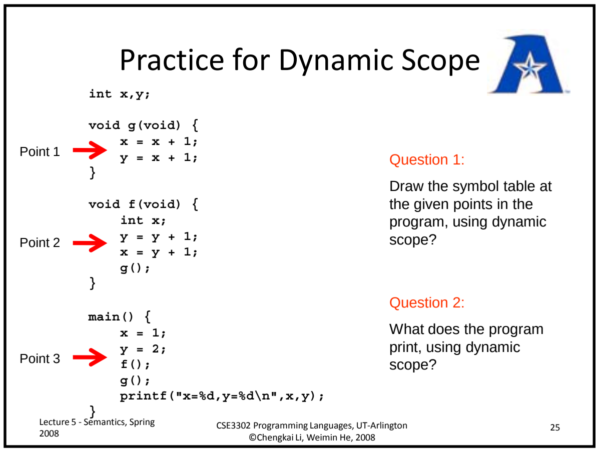
What if dynamic scope is used?

```
int x = 1;
char y = 'a';

void p(void) {
   double x=2.5;
   printf("%c\n",y);
}

void q(void) {
   int y = 42;
   printf("%d\n",x);
   p();
}

main() {
   char x = 'b';
   q();
}
```

The symbol table in **p**:
the bindings available in **p**

```
98

*
```

x
double, *2.5*, local to **p**
character, *'b'*, local to **main**
integer, *1*, global
y
integer, *42*, local to **q**
character, *'a'*, global



Practice for Dynamic Scope

```
            int x,y;

            void g(void) {
Point 1 →       x = x + 1;
                y = x + 1;
            }

            void f(void) {
                int x;
Point 2 →       y = y + 1;
                x = y + 1;
                g();
            }

            main() {
                x = 1;
Point 3 →       y = 2;
                f();
                g();
                printf("x=%d,y=%d\n",x,y);
            }
```

Question 1:

Draw the symbol table at the given points in the program, using dynamic scope?

Question 2:

What does the program print, using dynamic scope?

Lecture 5 - Semantics, Spring 2008

CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, Weimin He, 2008

25