Names

- Names: identify language entities
  - variables, procedures, functions, constants, data types, ...
- Attributes: properties of names
- Examples of attributes:
  - Data type:
    ```
    int n = 5;
    ```
  - 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
  - Dynamic binding (dynamic attribute):
    - occurs during execution

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
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;
    . . .
}
```

Access Hidden Declarations

- **scope resolution operator ::**: (C++)

```
int x;
void p(void) {
    char x;
    x = "a";
    . . .
}
main() {
    x = 2;
    . . .
}
```

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.

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

```c
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`

```plaintext
x integer, global
y character, global
```

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

```plaintext
x integer, global
y integer, local to `q`
```

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

```plaintext
x integer, local to `main`
```

Static Scope
Static Scope

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

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

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();
}
void q(void) {
    int y = 42;
    printf("%d\n",x);
p();
}
main() {
    x = 1;
    y = 2;
    f();
g();
printf("x=%d,y=%d\n",x,y);
}
```

• E.g., semantics of q

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

```c
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);
}
```

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?

Point 1
Point 2
Point 3