Declarations (1)

Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

First…

- Homework 3 will be announced in next week
- Final Project

Outline

- Declarations
- Declaration Syntax

Declaration

- Declarations play a important role in C programming.
- Declarations provide information to the compiler about the meaning of identifiers.
- In previous chapters, we have seen many examples of declarations without explaining them in detail.
- Let’s check them now!

```
// variable declaration
int i;

//function declaration
float f(float);
```

Declaration

- By the following declarations, we inform the compiler that:
- `int i;` in the current scop, the name `i` represents a variable of type `int`
- `float f(float);` `f` is a function that return float value and has one argument, also type `float`

```
// variable declaration
int i;

//function declaration
float f(float);
```

Declaration Syntax

- General form of a declaration:

declaration-specifiers declarators ;

- ***Declaration specifiers***
  - describe the properties of the variables or functions being declared.
- ***Declarators***
  - give their **names** and may provide **additional information** about their properties.

Declaration Syntax

<u>declaration-specifiers</u> declarators ;

```
// variable declaration
int i;
const float j[10];
char *p;

//function declaration
float f(float);
void f_2(char, int);
```

Please underscore the declaration-spcifiers in these examples.

Declaration

declaration-specifiers declarators ;

```
int i;
const float j[10];
```

- **Declaration specifiers** fall into three categories:
  - **Storage** classes
  - Type **qualifiers**
  - Type **specifiers**
- C99 has a fourth category, ***function specifiers,*** which are used only in function declarations (will introduce later).
- Type **qualifiers** and type **specifiers** should **follow** the **storage** class, but there are no other restrictions on their order.

storage-classes type-qualitifers/type-specifiers declarators;

declaration-specifiers declarators ;

Declaration Syntax

- There are four ***storage classes:*** `auto`, `static`, `extern`, and `register`.
- At most **one storage** class may appear in a declaration; if present, it should come first.
- There are three ***type qualifiers:*** `const` and `volatile, restrict` (C99, only for pointer variable)
- A declaration may contain **zero** or **more** type **qualifiers**.

```
static int i;
static extern int j; //WRONG! Only 1 storage class
extern const restrict int *k;// 2 type qualifiers, Ok!
```

declaration-specifiers declarators ;

Declaration Syntax

- The keywords `void`, `char`, `short`, `int`, `long`, `float`, `double`, `signed`, and `unsigned` are all ***type specifiers.***
- The order in which they are combined doesn't matter.
  - `int unsigned long` is the same as `long unsigned int`.
- **Type specifiers** also include specifications of structures, unions, and enumerations.
  - `struct point { int x, y; };`
- `typedef` names are also type specifiers.

declaration-specifiers declarators ;

Declaration Syntax

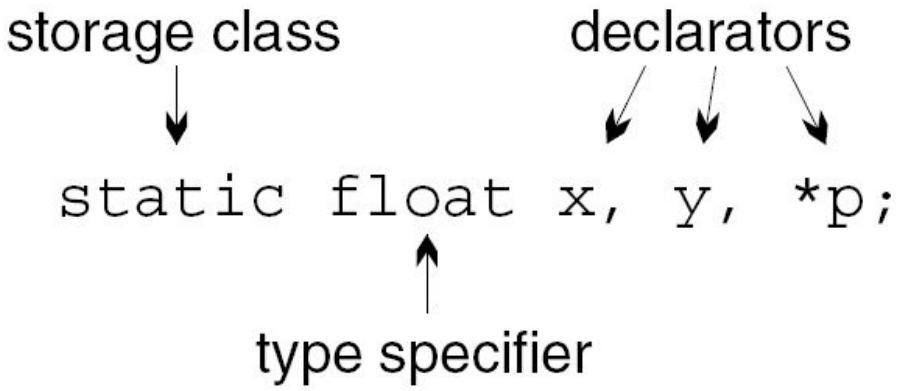
- Declarators include:
  - Identifiers (names of simple variables)
  - Identifiers followed by `[]` (array names)
  - Identifiers preceded by `*` (pointer names)
  - Identifiers followed by `()` (function names)
- Declarators are separated by commas.
- A declarator that represents a variable may be followed by an initializer.

```
int i[10];
int *p;
float f(float);
```

```
int i, j; //separated by commas
int *p = NULL; //initializer
```

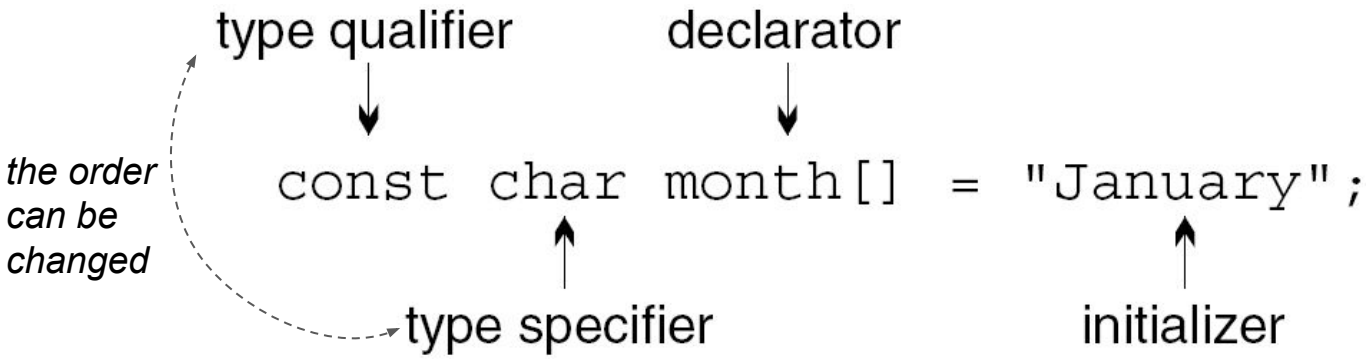
Declaration Syntax

- Let's look at some examples that illustrate these rules.
- A declaration with a storage class and three declarators:



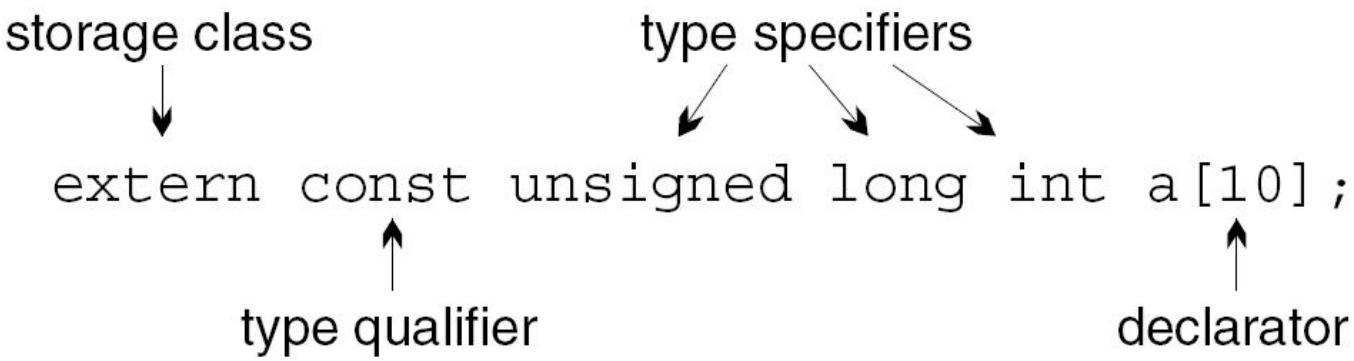
Declaration Syntax

- A declaration with a type qualifier and initializer but no storage class



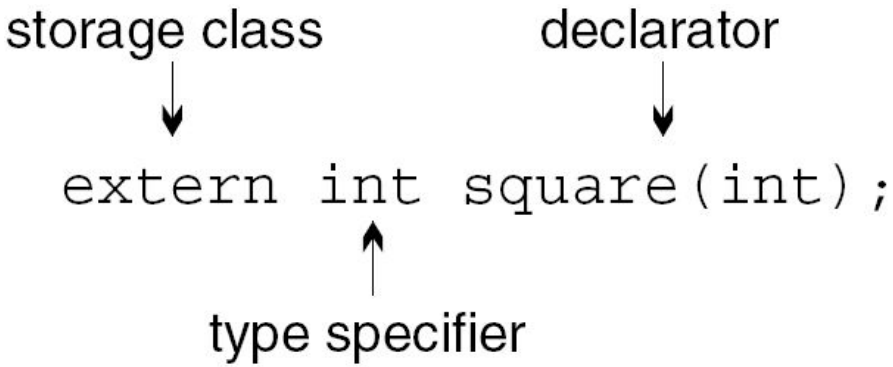
Declaration Syntax

- A declaration with a storage class, a type qualifier, and three type specifiers:



Declaration Syntax

- Function declarations may have a storage class, type qualifiers, and type specifiers:



Declaration Syntax

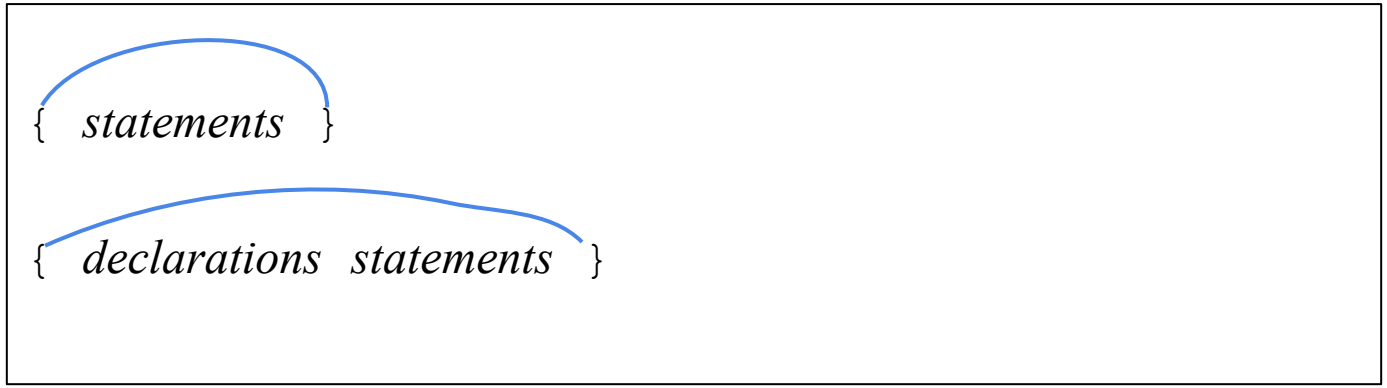
- Next, we will introduce the following topics one by one:
- Storage Classes (today)
- Type Qualifiers
- Declarators
- Intializers

Storage Classes

- Storage classes can be specified for variables, functions and parameters.
  - But we will concentrate on variables for now.
- There are four ***storage classes***
  - `auto`
  - `static`
  - `extern` (Writing Large Programs (2))
  - `register`

Blocks

- Before we introduce the storage classes, let's recall that the term *block* refers to the body of a function (the part in **braces**) or the selection statements (`if` and `switch`) and iteration statements (`for` and `while`) enclosed in braces `{ }`.



Blocks

- The variable has block scope; it can't be referenced outside the block.

```c
#include <stdio.h>

int main()
{

    int i = 2, j = 1;

    if (i > j) {
      /* swap values of i and j */
      int temp = i;
      i = j;
      j = temp;
    }

    printf("%d", temp);

    return 0;
}
```

```
main.c:23:18: error: 'temp' undeclared (first use in this function)
   23 |     printf("%d", temp);
      |                  ^~~~
```

Properties of Variables

- Every variable in a C program has three properties:
  - Storage duration
  - Scope
  - Linkage

Storage Duration

- The ***storage duration*** of a variable determines **when** memory is **set aside** for the variable and **when** that memory is **released**.
  - ***Automatic storage duration:*** Memory for variable is allocated when the surrounding **block** is **executed** and **deallocated** when the block **terminates**.
  - ***Static storage duration:*** Variable **stays at the same storage location** as long as the program is running, allowing it to **retain** its value indefinitely.
- For example, **external/global** variables have static storage duration and **local** variables have automatic storage duration.

Scope

- The ***scope*** of a variable is the portion of the program text in which the variable can be referenced.
  - ***Block scope:*** Variable is visible from its point of declaration to the end of the enclosing block.
  - ***File scope:*** Variable is visible from its point of declaration to the end of the enclosing file.

Scope

- C's **scope rules** enable the programmer (and the compiler) to determine **which meaning** is relevant at a given **point** in the program.
- In this example, the identifier `i` has **four** different meanings
- Declaration 1 is file scope, the rests are block scope.

```
int i ;                /* Declaration 1 */

void f(int i )         /* Declaration 2 */
{
  i = 1;
}

void g(void)
{
  int i = 2;           /* Declaration 3 */

  if (i > 0) {
    int i ;            /* Declaration 4 */

    i = 3;
  }

  i = 4;
}

void h(void)
{
  i = 5;
}
```

Linkage

- The ***linkage*** of a variable determines the extent to which it can be shared.
  - ***External linkage:*** Variable may be shared by **several** (perhaps all) **files** in a program.
  - ***Internal linkage:*** Variable is restricted to a single file but may be shared by the functions **in that file.**
  - ***No linkage:*** Variable **belongs** to a **single function** and can't be shared at all.

Properties of Variables

- The default storage duration, scope, and linkage of a variable depend on where it's declared:
  - Variables declared ***inside*** a **block** (including a function body) have *automatic* storage duration, *block* scope, and *no* linkage.
  - Variables declared ***outside*** **any block**, at the **outermost level** of a program, have *static* storage duration, *file* scope, and *external* linkage.

Properties of Variables

- We can alter these properties by specifying an explicit storage class: `auto`, `static`, `extern`, or `register`.

```
int i;
```

static storage duration
file scope
external linkage

```
void f(void)
{
  int j;
}
```

automatic storage duration
block scope
no linkage

The `auto` Storage Class

- The `auto` storage class is **legal only for variables** that belong to a **block**.
- The `auto` storage class is almost never specified explicitly, since it's the default for variables declared inside a block.
  - automatic storage duration, block scope, and no linkage

The `static` Storage Class

- The `static` storage class can be used with all variables, regardless of where they're declared.
- But it has a **different effect** regarding to where the variable is declared:
  - When used ***outside*** a block, `static` specifies that a variable has **internal linkage** (cannot shared with other files).
  - When used ***inside*** a block, `static` changes the variable's **storage duration** from **automatic** to **static**.

The `static` Storage Class

```
static int i;
```

static storage duration
file scope
internal linkage

```
void f(void)
{
  static int j;
}
```

static storage duration
block scope
no linkage

The `static` Storage Class

- A `static` variable declared within a block resides at the same storage location and retain same value throughout program execution.
- Properties of `static` variables:
  - A `static` variable is **initialized only once,** prior to program execution.
  - A `static` variable declared inside a function is shared by all calls of the function, including recursive calls.
  - A function may return a pointer to a `static` variable.

The `static` Storage Class

- Declaring a local variable to be `static` allows a function to retain information between calls.
- More often, we'll use `static` for reasons of efficiency
- Declaring `hex_chars` to be `static` saves time, because `static` variables are initialized only once.

```
char digit_to_hex_char(int digit){
      static const char hex_chars[16] = "0123456789ABCDEF";
      return hex_chars[digit];
}
```

The `register` Storage Class

- Using the `register` storage class in the declaration of a variable asks the compiler to store the variable in a register (high-speed storage in CPU).
- Mainly for program optimization
- But `register` is less used nowadays since most modern compliers are smart enough to automatically determine which variables should be stored in a register.
- Please read page 463 or the following article for more information.
  - https://www.geeksforgeeks.org/understanding-register-keyword/

declaration-specifiers declarators ;

Summary

- Declarations
- Declaration Syntax
  - General form of a declaration
- Review of Storage Duration, Block, Scope, and Linkage
- Storage Classes
  - `auto`
  - `static`
  - `extern` (Writing Large Programs (2))
  - `register`

storage-classes type-qualitifers/type-specifiers declarators;