Ch 18. notes – Declarations.

Declaration: declaration-specifier declarators.

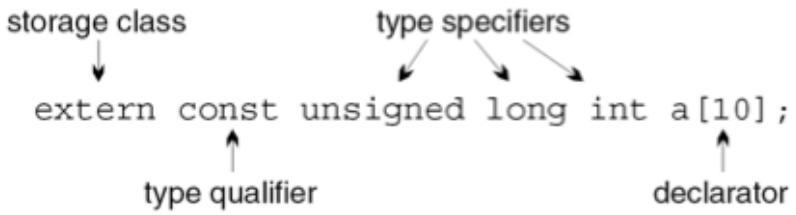
properties of the variable name & extra property.

declaration specifier (at most 1 and should come first in declaration)

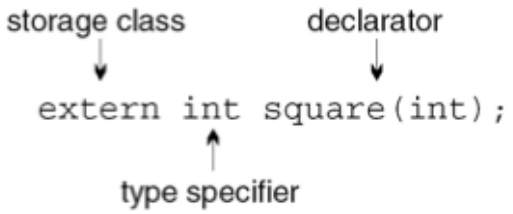
- Storage classes: auto static extern register
- Type quantifier: const volatile restrict (C99)
- Type specifier: void char short int long float double signed unsigned: order doesn't matter
- Function specifier (C99).

declarators: simple variable, [], *, ()

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



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



variable
- storage duration: automatic/static
- scope: block/file (in a block or in the whole file)
- linkage: external/internal/no linkage.

- 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.

```
int i;      static storage duration
            file scope
            external linkage

void f(void)
{
  int j;    automatic storage duration
            block scope
            no linkage
}
```

Static : external → internal

auto → static.

```
static int i;   static storage duration
                file scope
                internal linkage

void f(void)
{
  static int j; static storage duration
                block scope
                no linkage
}
```

static variables are initialized prior to program execution

The value store in static variable is shared in each calls.

definition: memory allocation, and possibly initialization. but it can only done once for each memory.

declaration: using "external", we can only declear a var without allocating memory for it.

All variables in an external declaration are always static.

- If the variable was declared `static` earlier in the file (outside of any function definition), then it has internal linkage.
- Otherwise (the normal case), the variable has external linkage.

- The `register` storage class is legal only for variables declared in a block.
- A `register` variable has the same storage duration, scope, and linkage as an `auto` variable.
- Since registers don't have addresses, it's illegal to use the `&` operator to take the address of a `register` variable.
- This restriction applies even if the compiler has elected to store the variable in memory.

it's in CPU rather than memory.

=> thus, register saves variables that are updates frequently.

Storage { external : allow to be call from other file
static: limited usage of the variable

No specified storage class => external link.

Static benefit: easier maintenance, future modification won't affect other files since is not accessible.
reduce namespace useage: name of static functions don't conflict with names in other files

Type qualifiers { const : read-only obj
volatile
restrict (C99) => pointer only.

Advantage of const:
- Serves as a form of documentation
- keep the value of the object unchanged.
- Alter the compiler that the object can be store in ROM (read-only memory)

# define => create a name for a numerical, char or string
const => creat name for any type of object.

a[]: brackets can be left empty if: 1) the array is a parameter.
2) it has an initializer 3) its storage class is extern
*: if it is a multidimension array, only the first brackets can be left empty.

Declaration rule:
1) read declaration from the inside out
2) favor [], () over *

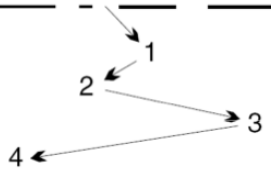
↙ pointer to a function that return a void type.

```
void (*pf)(int);
```

- Since `*pf` is enclosed in parentheses, `pf` must be a pointer.
- But `(*pf)` is followed by `(int)`, so `pf` must point to a function with an `int` argument.
- The word `void` represents the return type of this function.

- A second example of "zigzagging":

```
int *(*x[10])(void);
```



Type of x:
1. array of
2. pointers to
3. functions with no arguments
4. returning pointer to `int`

=> A function with no argument and return type of the array of pointers pointing to pointer to int

Certain things that can't be declared in C:
functions cannot return arrays.
Functions cannot return functions.
Arrays cannot store functions.
* in these cases, we can use pointers instead.

- Some programmers use type definitions to help simplify complex declarations.
- Suppose that `x` is declared as follows:

```
int *(*x[10])(void);
```

- The following type definitions make `x`'s type easier to understand:

```
typedef int *Fcn(void);
typedef Fcn *Fcn_ptr;
typedef Fcn_ptr Fcn_ptr_array[10];
Fcn_ptr_array x;
```

=> int *x(void)
=> int *(*x)(void).
=> int *(*x[10])(void)

- The initializer for a simple variable is an expression of the same type as the variable:

```
int i = 5 / 2;    /* i is initially 2 */
```

- If the types don't match, C converts the initializer using the same rules as for assignment:

```
int j = 5.5;      /* converted to 5 */
```

- The initializer for a pointer variable must be an expression of the same type or of type `void *`:

```
int *p = &i;
```

Review: int x=3; // int type variable x with value 3.

int* px= &x // pointer pointing at variable type int, with value of the address of x.

int** ppx=&px//pointer pointing at the pointer pointing at variable type int, with value of the address of px.

Initial value: automatic storage: no default initial value.

static storage: 0 by default./ base on its type