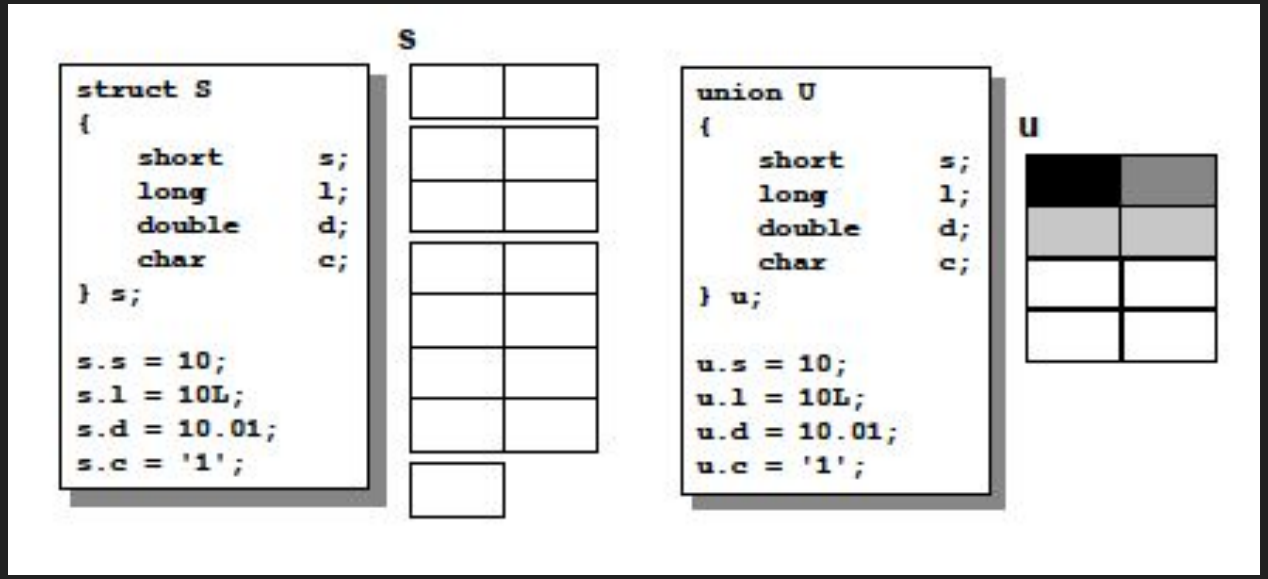
Miscellaneous things

Derrick

Union

A **`union`** is a variable which, at different times, may hold objects of different types and sizes.



Remembering

- It is up to the programmer to remember what type a union currently holds
- Unions are most often used in structs where a member stores a different type of data.

```
struct preprocessor_const
{
    char*           name;
    int             stored;
    union
    {
        long        lval;
        double      dval;
        char*       sval;
    } u;
};
```

```
#define  N_SIZE    10
#define  PI        3.1416
```

```
struct preprocessor_const s[10000];

s[0].name = "N_SIZE";
s[0].u.lval = 10L;
s[0].stored = STORED_LONG;

s[1].name = "PI";
s[1].u.dval = 3.1416;
s[1].stored = STORED_DOUBLE;
```

Enum

- Enumerated types provide an automated mechanism for generating named constants.

```
enum day { sun, mon, tue,
           wed, thu, fri, sat };

enum day today = sun;

if(today == mon)
    ....
```

```
#define sun     0
#define mon     1
#define tue     2
#define wed     3
#define thu     4
#define fri     5
#define sat     6

int today = sun;

if(today == mon)
    ....
```

Using different constants

- The constants used can be specified.

```
enum day { sun = 5, mon, tue, wed, thu, fri, sat };
enum direction { north = 0, east = 90, south = 180,
                                        west = 270 };
```

Working with large projects

- Large projects may potentially involve many hundreds of source files (modules).
- Global variables and functions in on module can be accessed in other modules.
- Global variables and functions can be specifically hidden inside a module
- Maintaining consistency between files can be a problem.

Storage classes

- **Scope** - Area where variable can be used.
- **Lifetime** - How long variable holds the memory.

Storage Classes

1. auto
2. **static**
3. **extern**
4. registers

Auto

- Scope - In the same block in which the variable has been declared. (local)
- Lifetime - only in the block in which the variable has been declared.
- Default value - garbage value

```
int x;

auto int x;
```

Static

- Scope - in the same block in which the variable has been declared (local)
- Lifetime - Until the completion of the program the variable will be alive.
- Default value - zero

```
static int i;
```

Extern

- Scope - throughout the program. (global)
- Lifetime - Until the completion of the program the variable will be alive.
- Default value - zero

```
extern int i;
```

Registers

- Stores the value in registers (CPU)
- Fast access during the run-time.
- Scope, Lifetime, Default value - same as **`auto`** (**default**)

```
register int i;
```

Example - 1

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

```
void increment()
{
    int a = 0; /* auto */
    a = a + 1;
    printf("a = %d\n", a);
}
```

Example - 2

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

```
void increment()
{
    static int a = 0;
    a = a + 1;
    printf("a = %d\n", a);
}
```

Example - 3

```
int main()
{
    int x = 3;
    extern int y;
    printf(“x = %d, y = %d\n”, x, y);
    return 0;
}
y = 10
```

static function (link)

- By default functions are implicitly declared as **`extern`**, which means they're accessible across the translation units (all files). But when use **`static`** it restricts visibility of the function to the translation unit in which it's defined.
- You can think of it as private function within the same file.

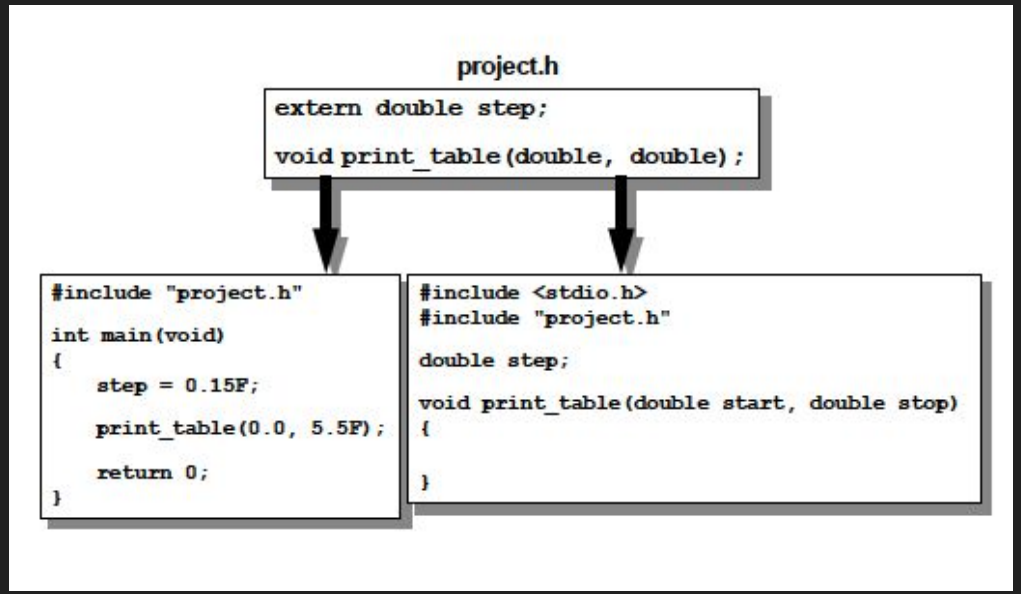
```
static void hello() {

    printf("Hello\n");

}
```

Use Header files

- Maintain consistency between modules by using header files.
- NEVER place an `extern` declaration in a module.
- NEVER place a prototype of a non `static` function in a module.



More resources about extern

- https://stackoverflow.com/questions/1433204/how-do-i-use-extern-to-share-variables-between-source-files
- https://www.geeksforgeeks.org/understanding-extern-keyword-in-c/

Extra

- **Get the preprocessor to declare the variables too!**

```
#if  defined(MAIN)
#define   EXTERN
#else
#define   EXTERN       extern
#endif

EXTERN double   step;
EXTERN long     current;
EXTERN short    res;
```

```
#define MAIN
#include "globals.h"
```

main.c

```
#include "globals.h"
```

first.c

```
#include "globals.h"
```

second.c