【C】Day4

Course	Advanced C
Study Date	@April 4, 2022

【Ch6】Structures

6.7 Typedef

C provides a facility called `typedef` for creating new data type names. For example, the declaration

```
typedef int Length;
```

makes the name Length a synonym for int.

6.8 Unions

A union is a variable that may hold objects of different types and sizes.

Unions provide a way to manipulate different kinds of data in a single area of storage.

Suppose that a constant may be an int, a float, or a character pointer.

The value of a particular constant must be stored in a variable of the proper type, yet it is most convenient for table management if the value occupies the same amount of storage and is stored in the same place regardless of its type.

This is the purpose of a union-a single variable that can legitimately hold any one of several types. The syntax is based on structures:

```
union u_tag {
  int ival;
  float fval;
  char *sval;
} u;
```

The variable `u` will be large enough to hold the largest of the three types; the specific size is implementation-dependent.

Syntactically, members of a union are accessed as

```
union-name.member
```

or

```
union-pointer->member
```

If the variable `utype` is used to keep track of the current type stored in `u`, then one might see code such as

```
if(utype == INT)
  printf("%d\n", u.ival);
else if(utype == FLOAT)
  printf("%f\n", u.fval);
else if(utype == STRING)
  printf("%s\n", u.sval);
else
  printf("bad type %d in utype\n", utype);
```

Unions may occur within structures and arrays, and vice versa. The notation for accessing a member of a union in a structure is identical to that for nested structures.

For example, in the structure array defined by

```
struct {
  char *name;
  int flags;
  int utype;
  union {
    int ival;
    float fval;
    char *sval;
  } u;
} symtab[NSYM];
```

The member `ival` is referred to as

```
symtab[i].u.ival
```

and the first character of the string `sval` by either of

```
*symtab[i].u.sval;
symtab[i].u.sval[0];
```

A union may only be initialized with a value of the type of its first member; thus the union `u` described above can only be initialized with an integer value.

```
union u {
  int ival;
  char *sval;
} a = {1}, b = {.sval = "hello"};

int main() {
  printf("%d\n", a.ival);
  printf("%s\n", b.sval);

  return 0;
}
```