Lecture 09: Structures Miscellaneous

Today’s Lecture

- ❑ Passing Structure Members as arguments to Function
- ❑ Passing Structure Variables as Parameters
- ❑ Returning Structure from Function
- ❑ Pointers to structure variables
- ❑ Passing Structure Pointers as Argument to a Function
- ❑ Returning a Structure Pointer from Function
- ❑ Passing Array of structures
- ❑ Dynamic Memory Allocation (DMA) of Structure Type Variables
- ❑ Struct vs Union

Union

- As structures, unions are also used to group a number of **different variables together**.
- The difference between union and structure is that, structure treat each of its member as a **different memory location** store in the main memory.
- While union treat each of its **member as a single memory location** store in the main memory.
  - i.e. all of the members of union share a common memory of union member.

Union

- Assume you are creating a program to record the quantity of different items, where quantity might be count, weight or volume.
- One approach is to use structures:

```
struct items
{
    int count;
    double weight;
    double volume;
};
struct items balls;
balls.count = 10;          //only use one member for one specific item;
```

- As we know balls quantity is measured using count. So, in this case, there is no need for weight and volume.

Union

- Similarly in the following statement:

```
struct items flour;
flour.weight = 10;   //only use one member for one specific item;
```

- As the quantity of flour is measured using weight. So, in this case, there is no need to store count and volume.
- A particular type of items at a time can be measured using only one of the quantity either a count or a weight or a volume.
- At this point our program has following limitations:
  - It takes more space than required, hence less efficient.
  - Someone might set more than one value.

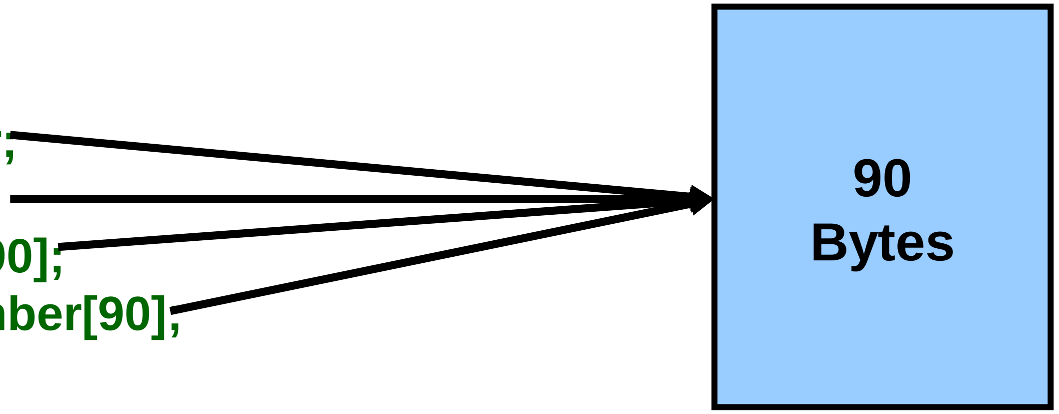
Union

- When a variable of type union is declared the compiler allocates memory sufficient to hold the largest member of the union.
- Since all members share the same memory you can only use one member of a union at a time, thus union is used to save memory.
- The syntax of declaring a union is as follows:

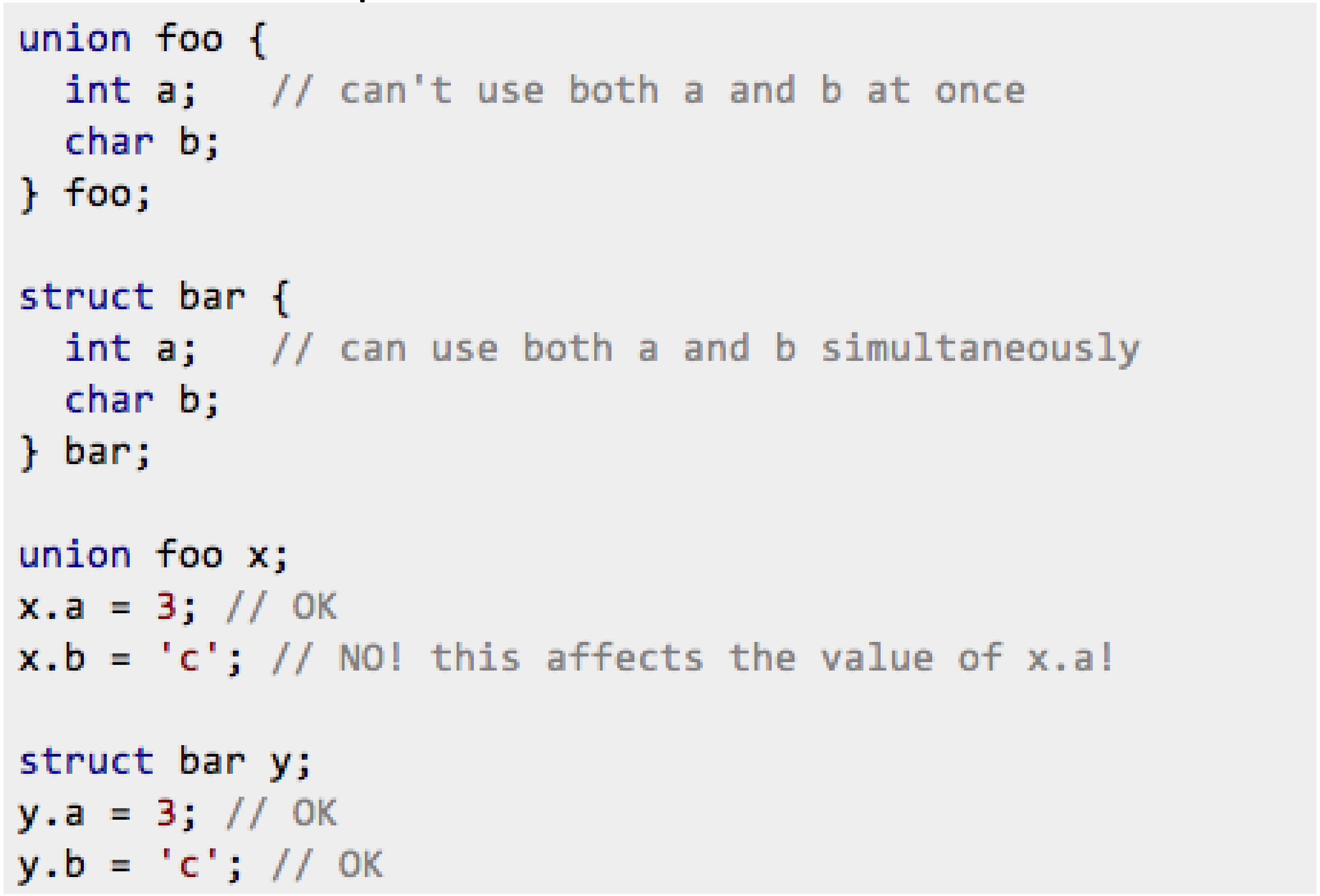
```
union tagname
{
    data_type member_1;
    data_type member_2;
    data_type member_3;
    ...
    data_type member_N;
};
```

Union Example

```
union searchOption
{
        int SearchByRollNumber;
        char SearchByName[90];
        char SearchByAddress[90];
       char SearchByPhoneNumber[90];
};
searchOption sv;
void main (void)
{
        int option = 0;
        switch (option)
        {
          case 0:  FunSearchRoll (sv.SearchByRollNumber); break;
         case 1:  FunSearchName(sv.SearchByName); break;
          case 2:  FunSearchByAddress(sv.SearchByAddress); break;
          case 3:  FunSearchByPhone(sv.SearchByPhoneNumber);
                   break;
        }
}
```



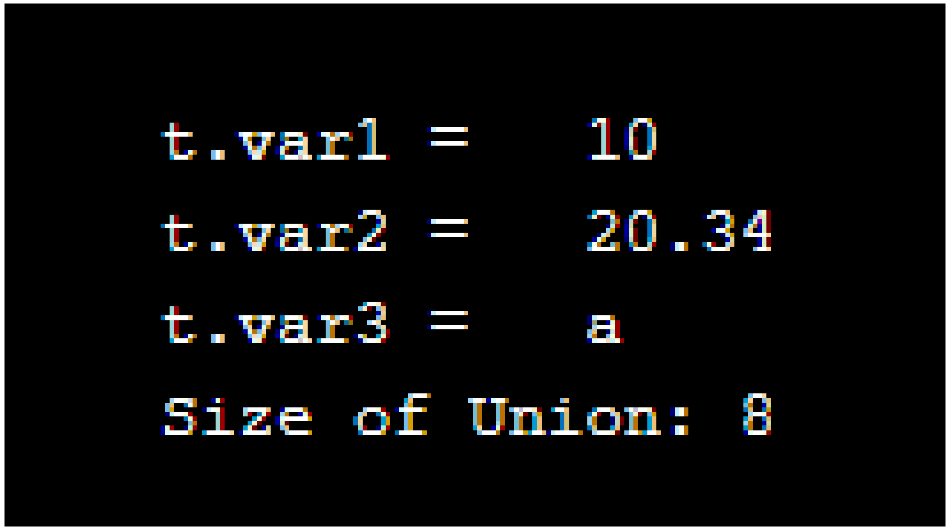
Union Example



```
union foo {
  int a;   // can't use both a and b at once
  char b;
} foo;

struct bar {
  int a;   // can use both a and b simultaneously
  char b;
} bar;

union foo x;
x.a = 3; // OK
x.b = 'c'; // NO! this affects the value of x.a!

struct bar y;
y.a = 3; // OK
y.b = 'c'; // OK
```

Union--Example

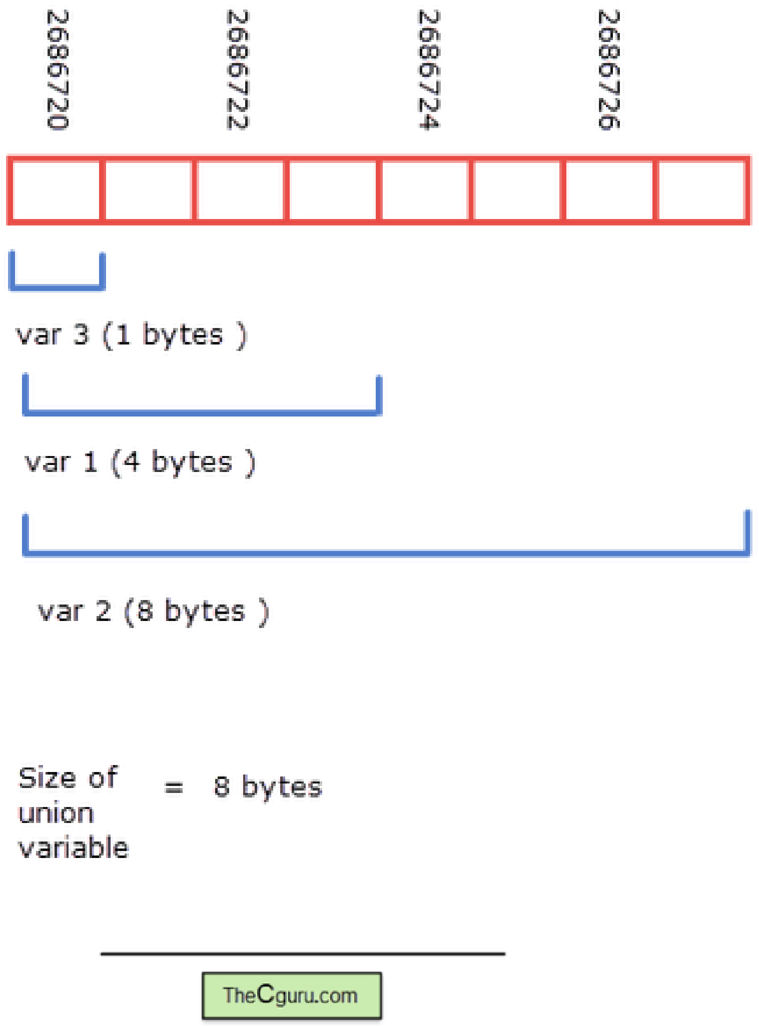
```
#include<iostream>
union data {
    int var1;
    double var2;
    char var3; };
int main() {
    union data t;
    t.var1 = 10;
    std::cout<<"t.var1 =   "<<t.var1<<std::endl;
    t.var2 = 20.34;
std::cout<<"t.var2 =   " <<t.var2<<std::endl;
    t.var3 = 'a';
std::cout<<"t.var3 =   " <<t.var3<<std::endl;
std::cout<<"Size of Union: "<<sizeof(t);
    return 0;
}
```



```
t.var1 =   10
t.var2 =   20.34
t.var3 =   a
Size of Union: 8
```

Size of Union Variable

```
union data

{
    int var1;
    double var2;
    char var3;

};
```



Union

- The primary purpose of a union is to save on memory.
- These are not used much in programming now as memory is cheap and other programming techniques are better to use.