Jiangxi University of Science and Technology

Chapter 12 Structures

Lecture1203 Union

12.5 Unions

- A union is a data type that reserves the same area in memory for two or more variables
  - **union**{
  - **char** key; **int** num;
  - **double** price;
  - } **val**;
  - Each of these types, but **only one at a time, can actually be assigned to the union variable**
  - A union reserves sufficient memory locations to accommodate its **largest member's data type**

12.5 Unions

- Individual union members are accessed using the same notation as structure members
- Typically, **a second variable keeps track of the current data type stored in the union**
  - **switch** (**uType**)
  - {
  - **case** ‘c’: printf("%c", val.key); **break**;
  - **case** ‘i’: printf("%d", val.num); **break**;
  - **case** ‘d’: printf("%f", val.price);**break**;
  - **default** : printf("Invalid type : %c", uType);
  - }

12.5 Unions

- **A type name** can be associated with a union to create templates
  - **union DateTime**
  - { **long** days; **double** time;};
  - **union DateTime** first, second, *pt;
- **Pointers** to unions use the same notation as pointers to structures

12.5 Unions

- Unions may be **members** of structures and arrays; structures, arrays, and pointers may be members of unions
  - struct{
  - char uType;
  - **union { char *text; double rate;}** uTax;
  - } flag;
  - rate is referenced as **flag.uTax.rate**
  - ***flag.uTax.text = ?**

Common Programming Errors

- Attempting to use structures and unions, as complete entities, in relational expressions
- Assigning an incorrect address to a pointer that is a member of a structure or union
- Storing one data type in a union and accessing it by the wrong variable name can result in an error that is particularly troublesome to locate

Common Compiler Errors

Error	Typical Unix-based Compiler Error Message	Typical Windows-based Compiler Error Message

Using the wrong type of braces when declaring a structure.
For example:
`struct`
`[`
`  int month;`
`  int day;`
`  int year;`
`] birth;`
	The following error will be reported on each line containing a brace:
	`(S) Syntax error.`
		`:error: syntax error : missing ';' before '[' : error: syntax error : missing ']' before ';'`

Attempting to initialize the elements of a structure inside the declaration.
For example:
`struct`
`{`
`  int month = 6;`
`  int day;`
`  int year;`
`} birth;`
	`S) Syntax error: possible missing ';' or ','?`	`:error: 'month' : only const static integral data members can be initialized inside a class or struct`

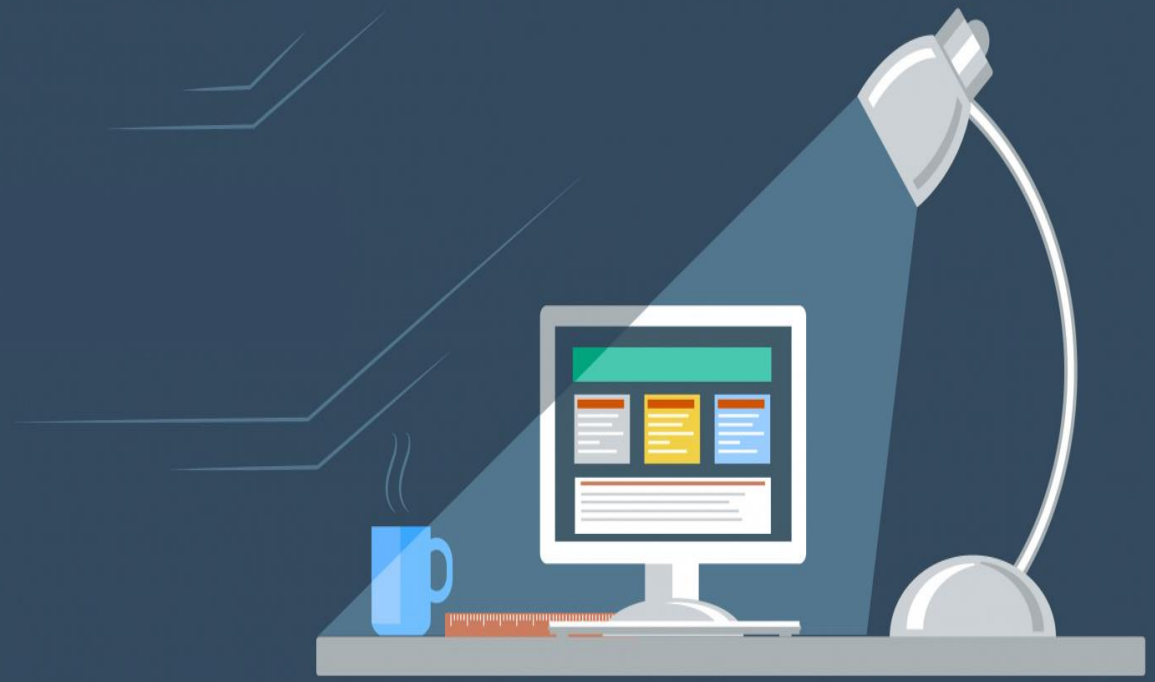
Assigning a pointer to a structure rather than the address of the structure.
For example:
`int main()`
`{`
`  struct Date *ptr;`
`  struct Date birth;`
`  ptr = birth;`
`}`
	`(S) Operation between types "struct Date*" and "struct Date" is not allowed.`	`:error: '=' : cannot convert from 'Date' to 'Date *'`

Summary

- A structure allows individual variables to be grouped under a common variable name
- A structure type name can be used to create a generalized structure type describing the form and arrangement of elements in a structure
- Structures are particularly useful as elements of arrays
- Individual members of a structure are passed to a function in the manner appropriate to the data type of the member being passed
- Structure members can be any valid C data type, including structures, unions, arrays, and pointers
- Unions are declared in the same manner as structures

Reference

- https://www.codesdope.com/blog/article/int-main-vs-void-main-vs-int-mainvoid-in-c-c/



Jiangxi University of Science and Technology

Unions in C Programming: Definition & Example

THE C PROGRAMMING LANGUAGE

Unions in C programming

THE C PROGRAMMING LANGUAGE

- **Union** is a data type in C programming that allows different data types to be stored in the same memory locations. Union provides an efficient way of reusing the memory location as only one of its members can be accessed at a time. A union is used almost in the same way you would declare and use a structure.

Syntax

```
1.union Record{
2. int i;
3. float f;
4. char str[10];
5.} record;
```

In this example, we initialized a union named **Record** which had three data members. A reference to this union is created which is called as **record**

Unions in C programming

THE C PROGRAMMING LANGUAGE

- A union, is a collection of variables of different types, just like a structure. However, with unions, you can only store information in one field at any one time
- You can picture a union as like a chunk of memory that is used to store variables of different types.
- Once a new value is assigned to a field, the existing data is wiped over with the new data.
- A union can also be viewed as a variable type that can contain many different variables (like a structure), but only actually holds one of them at a time (not like a structure).
- This can save memory if you have a group of data where only one of the types is used at a time.
- The size of a union is equal to the size of it's largest data or element

Memory Allocation

- In the above-declared union, the memory occupied by the union will be the memory required for the largest member of the union. In this case the union will occupy 10 bytes of memory space as the character string occupies the maximum space.
- The following shows the total memory size. If you print **sizeof(record)**, this will give you the size of the union; which in this case will be 10

Syntax

```
1.union Record{
2. int i;
3. float f;
4. char str[10];
5.} record;
```

How to create union variables?

- Union variables can be created in similar manner as structure variables

```
union car
{
   char name[50];
   int price;
} car1, car2, *car3;
```

OR

```
union car
{
   char name[50];
   int price;
};

int main()
{
   union car car1, car2, *car3;
   return 0;
}
```

Defining a structure and Union

THE C PROGRAMMING LANGUAGE

- A structure is a user-defined data type available in C that allows to combining data items of different kinds. Structures are used to represent a record.

Defining a structure: To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows:

```
struct [structure name]
{
    member definition;
    member definition;
    ...
    member definition;
};
```

union

THE C PROGRAMMING LANGUAGE

- A union is a special data type available in C that allows storing different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purposes.

Defining a Union: To define a union, you must use the **union** statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows:

Similarities between Structure and Union

THE C PROGRAMMING LANGUAGE

1. Both are user-defined data types used to store data of different types as a single unit.
2. Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.
3. Both structures and unions support only assignment = and sizeof operators. The two structures or unions in the assignment must have the same members and member types.
4. A structure or a union can be passed by value to functions and returned by value by functions. The argument must have the same type as the function parameter. A structure or union is passed by value just like a scalar variable as a corresponding parameter.
5. '.' operator is used for accessing members

Differences

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Example

- In this example, we will use the above mentioned union called *record*.
- We can access the data members of the union using the '.' operator.

```c
#include <stdio.h>
#include <string.h>
union Record {
  int i;
  float f;
  char str[20];
};
int main( ) {

  union Record record;

  record.i = 10;
  record.f = 220.5;
  strcpy( record.str, "C Programming");

  printf( "record.i : %d\n", record.i);
  printf( "record.f : %f\n", record.f);
  printf( "record.str : %s\n", record.str);

  return 0;
}
```

Output

```
record.i : 1917853763
record.f : 412236058032779486045275994368.000000
record.str : C Programming
```

- As we see from the output, the value of *record.i* and *record.f* is corrupted.
- This is because the final memory assignment was done to *record.str*, which is why on printing *record.str*, it gives the correct value.
- In order to print the correct values of each data member, it is better to use the *printf* statement one at a time as follows :

```
1.record.i = 10;
2.printf( 'record.i : %d\n', record.i);
3.record.f = 220.5;
4.printf( 'record.f : %f\n', record.f);
5.strcpy( record.str, 'C Programming');
6.printf( 'record.str : %s\n', record.str)
```

Difference between Unions and Structures

THE C PROGRAMMING LANGUAGE

- structure is better because as memory is shared in union ambiguity is more.

•Below table will help you how to form a C union, declare a union, initializing and accessing the members of the union.

Using normal variable	Using pointer variable

Syntax:
union tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
	Syntax:
	union tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};

Example:
union student
{
int mark;
char name[10];
float average;
};
	Example:
	union student
	{
	int mark;
	char name[10];
	float average;
	};

Declaring union using normal variable: union student report;	**Declaring union using pointer variable:** union student *report, rep;
Initializing union using normal variable: union student report = {100, “Mani”, 99.5};	**Initializing union using pointer variable:** union student rep = {100, “Mani”, 99.5}; report = &rep;
Accessing union members using normal variable: report.mark; report.name; report.average;	**Accessing union members using pointer variable:** report -> mark; report -> name; report -> average;

THE C PROGRAMMING LANGUAGE

江西理工大學 JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

DIFFERENCE BETWEEN STRUCTURE AND UNION IN C:

C Structure	C Union
Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
Structure occupies higher memory space.	Union occupies lower memory space over structure.
We can access all members of structure at a time.	We can access only one member of union at a time.

Structure example:
struct student
{
int mark;
char name[6];
double average;
};
	Union example:
	union student
	{
	int mark;
	char name[6];
	double average;
	};

For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes

THE C PROGRAMMING LANGUAGE

江西理工大學 JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
// C program to illustrate differences  between structure and Union
        #include <stdio.h>
        #include <string.h>
        // declaring structure
struct struct_example {
                int integer;
                float decimal;
                char name[20];
        };
// declaraing union
        union union_example {
                int integer;
                float decimal;
                char name[20];
};
void main() {
// creating variable for structure  and initializing values difference  six
                struct struct_example s={18,38,"geeksforgeeks"};

// creating variable for union  and initializing values
        union union_example u={18,38,"geeksforgeeks"};
                printf("structure data:\n integer: %d\n" "decimal: %.2f\n name: %s\n", s.integer, s.decimal, s.name);
                printf("\nunion data:\n integeer: %d\n""decimal: %.2f\n name: %s\n", u.integer, u.decimal, u.name);
// difference two and three
        printf("\nsizeof structure : %d\n", sizeof(s));
        printf("sizeof union : %d\n", sizeof(u));
```

```
// difference five
    printf("\n Accessing all members at a time:");
    s.integer = 183;
    s.decimal = 90;
    strcpy(s.name, "geeksforgeeks");
    printf("structure data:\n integer: %d\n ""decimal: %.2f\n name: %s\n",s.integer, s.decimal, s.name);
    u.integer = 183;
    u.decimal = 90;
    strcpy(u.name, "geeksforgeeks");
    printf("\nunion data:\n integeer: %d\n ""decimal: %.2f\n name: %s\n",u.integer, u.decimal, u.name);
    printf("\n Accessing one member at time:");
    printf("\nstructure data:");
    s.integer = 240;
```

```
printf("\ninteger: %d", s.integer);
        s.decimal = 120;
        printf("\ndecimal: %f", s.decimal);
        strcpy(s.name, "C programming");
        printf("\nname: %s\n", s.name);
        printf("\n union data:");
        u.integer = 240;
        printf("\ninteger: %d", u.integer);
        u.decimal = 120;
        printf("\ndecimal: %f", u.decimal);
        strcpy(u.name, "C programming");
        printf("\nname: %s\n", u.name);
//difference four
        printf("\nAltering a member value:\n");
        s.integer = 1218;
        printf("structure data:\n integer: %d\n "" decimal: %.2f\n name: %s\n", s.integer, s.decimal, s.name);
        u.integer = 1218;
        printf("union data:\n integer: %d\n"" decimal: %.2f\n name: %s\n", u.integer, u.decimal, u.name);
}
```

Output:

```
structure data:
integer: 18
 decimal: 38.00
 name: geeksforgeeks
 union data:
integeer: 18
 decimal: 0.00
name: ?
sizeof structure: 28
sizeof union: 20
 Accessing all members at a time: structure data:
integer: 183
 decimal: 90.00
 name: geeksforgeeks
union data:
 integeer: 1801807207
decimal: 27732287172115951000000000.00
 name: geeksforgeeks
Accessing one member at a time: structure data:
 integer: 240 decimal: 120.000000
name: C programming
union data: integer: 240
decimal: 120.000000
name: C programming
Altering a member value: structure data:
integer: 1218
 decimal: 120.00
name: C programming
union data:
integer: 1218
 decimal: 0.00
 name: ?
```

Reference

- BOOK
- Some part of this PPT given by Prof 欧阳城添

(Prof: Chengtian Ouyang)

- with special thank
- https://www.codingunit.com/c-tutorial-first-c-program-hello-world