C Programming

Lecture 7: struct, union and enum

Name	Gender	Age
Tom	Male	22
Jack	Male	21
Jane	Female	21

Lecturer: *Dr.* Wan-Lei Zhao

Autumn Semester 2022

Email: wlzhao@xmu.edu.cn,

Opening Discussion

- Given we have following informatio for *40* students
  1. student number
  2. name
  3. age
  4. gender
  5. height
  6. GPA
- We now want to build records for all the students

```
1 int main()
2 {
3     char std1nm[64], std2nm[64], ...;
4     char std1nb[11], std2nb[11], ...;
5     int  std1ag, std2ag, ...;
6     char std1gd[5], std1gd[5], ...;
7     ...
8 }
```

Outline

Composite Data Types

- It is valid/OK to do it in the way we learned
- However, it is not convenient
- C provides us the way to extend current data types
- We can combine primitive types into one type

```
1 struct STD {
2     char stdNm[64];
3     char stdNb[11];
4     int  age;
5     char gender[5];
6 };
```

- "struct STD" is a new data type
- Its role is similar as int, or float,...

struct: grammar of definition

```
struct structTag {
  type1 member1;
  type2 member2;
  ...
  typeN memberN; };
```

- Keyword "struct" is required, it tells C you are going to define a composite type
- structTag gives a **unique** tag for this new type
- You list all the memebers and their corresponding types
- ";" is required at the end
- Keep in your mind, you define a **type** instead of a variable/constant

struct: define variable of composite type (1)

struct structTag **record**;

- Keyword "struct" and structTag are required
- "record" is the variable name of structTag type

struct: define variable of composite type (2)

struct structTag **record**;

- Keyword "struct" and structTag are required
- "record" is the variable name of structTag type

```
1  struct STD {
2      char stdNm[64];
3      char stdNb[11];
4      int  age;
5      char gender[5];
6  };
7  int main()
8  {
9      struct STD record;
10     struct STD stds[40];
11 }
```

struct: initialize variable of composite type (1)

- Each memember in the composite type variable is treated as a variable
- They are visited via "var.member1"

```
1  struct STD {
2      char stdNm[64];
3      char stdNb[11];
4      int  age;
5      char gender[5];
6  };
7  int main()
8  {
9      struct STD record;
10     strcpy(record.stdNm, "Min-Li");
11     strcpy(record.stdNb, "11201522031");
12     record.age = 20;
13     strcpy(record.gender, "male");
14 }
```

struct: initialize variable of composite type (2)

```c
#include <stdio.h>
#include <string.h>
struct STD {
    char stdNm[64];
    char stdNb[11];
    int  age;
    char gender[5];};
int main()
{
    struct STD std;
    strcpy(std.stdNm, "Min-Li");
    strcpy(std.stdNb, "22031");
    std.age = 20;
    strcpy(std.gender, "male");
    printf("Name:-%s\n", std.stdNm);
    printf("Numb:-%s\n", std.stdNb);
    printf("Age:-%d\n", std.age);
    printf("Gender:-%s\n", std.gender);
    return 0;
}
```

struct: exmaple (1)

- Please build a struct type for date (Year, month and day)
- Work out which day it is of the year
  1. We need struct type to keep date inform
  2. We need to calculate which day of the year is
  3. It depends on year (whether it is a leap year)
  4. Depends on the month
  5. Depends on the date

5 minutes to think about it...

struct: exmaple (2)

[General procedure]

1. Accept input, save the information to a date structure
2. Check whether the year is leap year or not
3. Check which month it is
4. We need an array to keep the days of months

struct: exmaple (3)

[General procedure in more detail]

1. Define a date struct
2. Accept input, save the information to a date structure
3. Initialize of days of months (12 months)
4. If it is leap year and date.month >= 3
5. Plus 1 day to the total
6. End-If
7. For i from 1 to (date.month-1)
8. sum up days of months before current month
9. End-for

struct: exmaple (4)

```c
struct DATE {
    int day, month, year;
};

int main()
{
    struct DATE date;
    int dyMonth[]={31,28,31,
    30,31,30,31,31,
    30,31,30,31};
    int i = 1, dayth = 0;
    printf("Year:");
    scanf("%d", &date.year);
    printf("Month:");
    scanf("%d", &date.month);
    printf("Day:");
    scanf("%d", &date.day);
    if(isLeap(date.year))
    {
        dayth += 1;
    }
    for(;i<date.month;i++)
    {
        dayth+= dyMonth[i-1];
    }
    dayth += date.day;
    return 0;
}
```

Is there anything wrong?? Two mistakes!!

struct: exmaple (5)

```c
struct DATE {
    int day, month, year;
};
int isLeap(int year)
{   if(year%4==0) {
        if(year%400==0){
            return 1;
        }else if(year%100=0){
            return 0;
        }
        return 1;
    }else{
        return 0;
    }//end-if-else
}//end-isLeap
int main()
{   struct DATE date;
    int dyMonth[]={31,28,31,
    30,31,30,31,31,
    30,31,30,31};
    int i = 1, dayth = 0;
    printf("Year:");
    scanf("%d", &date.year);
    printf("Month:");
    scanf("%d", &date.month);
    printf("Day:");
    scanf("%d", &date.day);
    if(isLeap(date.year)&&date
     .month>2)
    {
        dayth += 1;
    }
    for(;i<date.month;i++)
    {
        dayth+= dyMonth[i-1];
    }
    dayth += date.day;
    return 0;
}
```

struct: size of the struct type (1)

- Now let's consider another problem
- What is the size (bytes occupied) of struct type variable

```c
struct DATE {
    int day, month, year;
};
struct STD{
    char Name[10];
    int age;
    char gender[6];
};
int main()
{
    printf("%d\n", sizeof(
     struct DATE));
    printf("%d\n", sizeof(
     struct STD));
    return 0;
}
```

[Output]

```
12
24
```

- Can you figure out why?

struct: size of the struct type (2)

- Now let's consider another problem
- What is the size (bytes occupied) of struct type variable

```c
struct DATE {
    int day, month, year;
};
struct STD{
    char Name[10];
    int age;
    char gender[6];
};
int main()
{
    printf("%d\n", sizeof(
     struct DATE));
    printf("%d\n", sizeof(
     struct STD));
    return 0;
}
```

[Output]

```
12
24
```

- Name will be given with 12 bytes instead of 10
- gender will be given with 8 bytes instead of 6
- For the convevience of memory allocation
- This could be different from one compiler to another

struct example: complex number (1)

- Given two complex number a = 2+3i and b = 4-i
- You are asked to define a struct of Complex
- Fulfill c = a+b

Think about it in 5 minutes ...

struct example: complex number (2)

- Define the struct

```
1 struct Complex {
2     float real, virt;
3 };
```

struct example: complex number (3)

```
1 struct Complex {
2 float real, virt;
3 };
4
5 struct Complex add( struct Complex a, struct Complex b){
6     //fill by yourself
7 }
```

struct example: complex number (4)

```
1  struct Complex {
2  float real, virt;
3  };
4
5  struct Complex add( struct Complex a, struct Complex b){
6      struct Complex c;
7      c.real = a.real + b.real;
8      c.virt  = a.virt + b.virt;
9      return c;
10 }
11
12 int main(){
13  struct Complex a = {2,3} , c;
14  struct Complex b = {4,-1};
15  c = add(a, b);
16  printf("c=%f+%fi", c.real, c.virt);
17  return 0;
18 }
```

struct: typedef to save code (1)

- In "struct STD", "struct" has been repeated everywhere
- We can use "typedef" to save up our typing

```
1  struct DATE {
2      int day, month, year;};
3  struct STD{
4      char Name[10];
5      int age;
6      char gender[6];};
7  typedef struct STD StdType;
8  typedef struct DATE DatType;
9  int main()
10 {
11     DatType date;
12     StdType std;
13     printf("%d\n", sizeof(DatType));
14     ...
15 }
```

- During compiling stage
- "StdType" is replaced by "struct STD"

struct: typedef to save code (2)

- You can apply typedef to any type

```c
#include <stdio.h>
typedef unsigned int uint;

int main()
{
    uint a = 32768;
    printf("%d\n", a);
    printf("%d\n", sizeof(uint));
    return 0;
}
```

- During compiling stage
- "uint" is replaced by "unsigned int"
- You actually give a **nickname** to the type by typedef

Outline

union

- Sometimes it is not necessary to reserve a field for each struct member
- Several fields are allowed to share the same block of memory
- This special type of structure is called union

```
1 struct Data {
2     short i;
3     float f;
4     char str[20];
5 };
```

```
1 union Data {
2     short i;
3     float f;
4     char str[20];
5 };
```

union: definition (1)

```
union [union tag] {
  type1 member1;
  type2 member2;
  ...
};
```

- It is basically very similar as struct
- However, the members are kept in different way

```c
struct Data1 {
    short i;
    float f;
    char str[20];};
```

```c
union Data2 {
    short i;
    float f;
    char str[20];
};
```

union: definition (2)

```
1  struct Data1 {
2      short i;
3      float f;
4      char str[10];};
5
6  union Data2 {
7      short i;
8      float f;
9      char str[10];};
10 int main()
11 {
12     Data1 d1;
13     Data2 d2;
14     printf("Size-of-d1-%d", sizeof(d1));
15     printf("Size-of-d2-%d", sizeof(d2));
16     return 0;
17 }
```

[Output:???]

union: definition (3)

```c
int main()
{
    Data1 d1;
    Data2 d2;
    printf("Size-of-d1-%d", sizeof(d1));
    printf("Size-of-d2-%d", sizeof(d2));
    return 0;
}
```

Size of d1: 20
Size of d2: 12

- Can you figure out why??

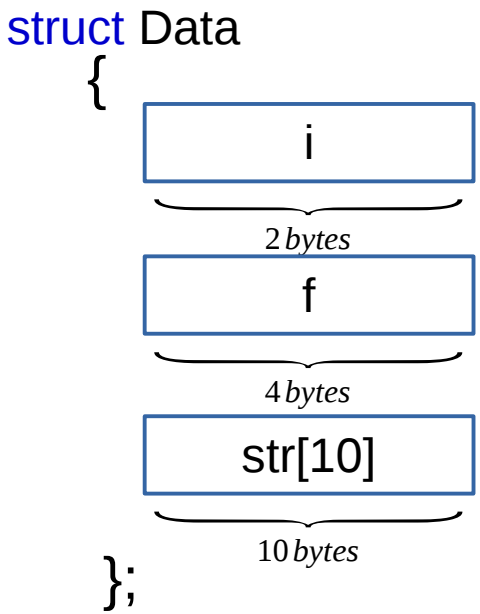
union: definition (4)

```
1 int main()
2 {
3     Data1 d1;
4     Data2 d2;
5     printf("Size-of-d1-%d", sizeof(d1));
6     printf("Size-of-d2-%d", sizeof(d2));
7     return 0;
8 }
```

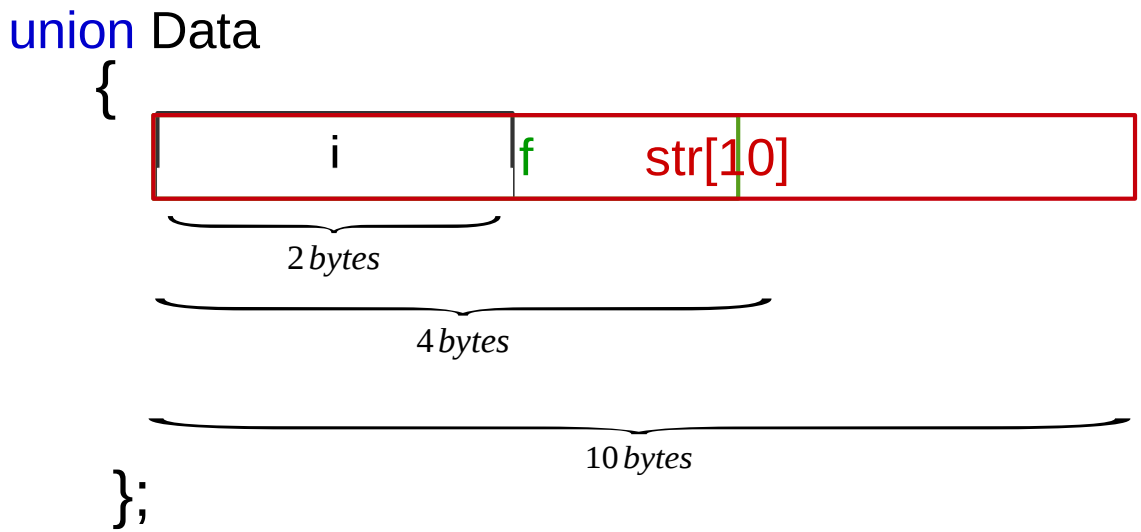
Size of d1: 20
Size of d2: 12

- For the convenience of memory allocation
- str will be given 12 bytes instead of 10

union: how they are kept in the memory



(a) struct



(b) union

union: learn by example (1)

```c
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];};
int main()
{
    union Data data;
    data.i = 10;
    data.f = 220.5;
    strcpy( data.str, "C-Programming");

    printf( "data.i-:-%d\n", data.i);
    printf( "data.f-:-%f\n", data.f);
    printf( "data.str-:-%s\n", data.str);
    return 0;
}
```

- See what the output??

union: learn by example (2)

```c
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];};
int main(){
    union Data data;
    data.i = 10;
    data.f = 220.5;
    strcpy( data.str, "C-Programming");
    printf( "data.i-:-%d\n", data.i);
    printf( "data.f-:-%f\n", data.f);
    printf( "data.str-:-%s\n", data.str);
    return 0;
}
```

data.i : 1917853763

data.f : 4122360580327794860452759994368.000000

data.str : C Programming

union: learn by example (3)

```c
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];};
int main(){
    union Data data;
    data.i = 10;
    strcpy( data.str, "C-Programming");
    data.f = 220.5;
    printf( "data.i-:-%d\n", data.i);
    printf( "data.f-:-%f\n", data.f);
    printf( "data.str-:-%s\n", data.str);
    return 0;
}
```

union: learn by example (4)

```c
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];};
int main(){
    union Data data;
    data.i = 10;
    strcpy( data.str, "C-Programming");
    data.f = 220.5;
    printf( "data.i-:-%d\n", data.i);
    printf( "data.f-:-%f\n", data.f);
    printf( "data.str-:-%s\n", data.str);
    return 0;
}
```

data.i : 1130135552
data.f : 220.500000
data.str :

union: learn by example (5)

```c
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];};
int main(){
    data.i = 10;
    printf( "data.i-:-%d\n", data.i);
    data.f = 220.5;
    printf( "data.f-:-%f\n", data.f);
    strcpy( data.str, "C-Programming");
    printf( "data.str-:-%s\n", data.str);
    return 0;
}
```

union: learn by example (6)

```c
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];};
int main(){
    data.i = 10;
    printf( "data.i-:-%d\n", data.i);
    data.f = 220.5;
    printf( "data.f-:-%f\n", data.f);
    strcpy( data.str, "C-Programming");
    printf( "data.str-:-%s\n", data.str);
    return 0;
}
```

data.i : 10
data.f : 220.500000
data.str : C Programming

Outline

enum: motivation

- Sometimes, we feel it is more meaningful
- with symbols: Janurary, Feburary ,..., December
- than numbers: 1, 2, ..., 12
- enum allows us to do a kind of correlating
- Numbers are assigned with readable symbols

enum: definition (1)

```
enum enumName{memb1, memb2, memb3,...};
```

- You enumerate all the members' name inside "{}"
- They are symbols
- They will be related to integer 0, 1, 2,... automatically

enum: definition (2)

enum enumName{memb1, memb2, memb3};

```
1 enum Month {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
      Nov, Dec};
2 int main()
3 {
4     ...
5 }
```

- You enumerate all the members' name inside "{}"
- They are symbols
- They will be related to integer 0, 1, 2,... automatically

enum: how to use it

```c
#include <stdio.h>
enum Month {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
      Nov, Dec};
int main()
{
    enum Month m;
    m = Feb;
    printf("Month is: %d\n", m);
    return 0;
}
```

[Output]

```
Month is: 1
```

- **Feb** is a symbol instead of a string
- They will be related to integer 0, 1, 2,... automatically

enum: learn by example (1)

```c
#include <stdio.h>
enum Week {Mon=1, Tue=1, Wed=3,
 Thu=5, Fri, Sat=4, Sun};
int main()
{
    enum Week wk;
    wk=Wed;
    printf("Wed: %d\n", wk);
    wk=Fri;
    printf("Fri: %d\n", wk);
    wk=Sun;
    printf("Sun: %d\n", wk);
    return 0;
}
```

[Output]

```
Wed: 3
Fri: 5
Sun: 6
```

enum: learn by example (2)

```c
#include <stdio.h>
enum Week {Mon=1, Tue=1, Wed=3,
 Thu=5, Fri, Sat=4, Sun};
int main()
{
    enum Week wk;
    wk=Wed;
    printf("Wed:-%d\n", wk);
    wk=Fri;
    printf("Fri:-%d\n", wk);
    wk=Sun;
    printf("Sun:-%d\n", wk);
    return 0;
}
```

[Output]

```
Wed: 3
Fri: 6
Sun: 5
```

- Can you figure out why??
- This way is valid, but NOT suggested

enum: learn by example (2)

```c
#include <stdio.h>
enum Week {Mon=1, Tue, Wed,
 Thu, Fri, Sat, Sun};
int main()
{
    enum Week wk;
    wk=Wed;
    printf("Wed: %d\n", wk);
    wk=Fri;
    printf("Fri: %d\n", wk);
    wk=Sun;
    printf("Sun: %d\n", wk);
    return 0;
}
```

[Output]

```
Wed: 3
Fri: 5
Sun: 7
```

- This is the right way

enum: learn by example (3)

```c
#include <stdio.h>
enum Week {Mon=1, Tue, Wed,
 Thu, Fri, Sat, Sun};
typedef enum Week WkType;
int main()
{
    WkType wk;
    wk=Wed;
    printf("Wed: %d\n", wk);
    wk=Fri;
    printf("Fri: %d\n", wk);
    wk=Sun;
    printf("Sun: %d\n", wk);
    return 0;
}
```

[Output]

```
Wed: 3
Fri: 5
Sun: 7
```

- You can use "typedef" to save up your coding efforts