**Introduction to C Programming**

**Lecture 7: self-defined types**

**Wenjin Wang** [**wangwj3@sustech.edu.cn**](mailto:wangwangj3@sustech.edu.cn)

**10-21-2022**

### Course syllabus

|  |  |  |
| --- | --- | --- |
| **Nr.** | **Lecture** | **Date** |
| **1** | **Introduction** | **2022.9.9** |
| **2** | **Basics** | **2022.9.16** |
| **3** | **Decision and looping** | **2022.9.23** |
| **4** | **Array & string** | **2022.9.30** |
| **5** | **Functions** | **2022.10.9 (补)** |
| **6** | **Pointer** | **2022.10.14** |
| **7** | **Self-defined types** | **2022.10.21** |
| **8** | **I/O** | **2022.10.28** |

|  |  |  |
| --- | --- | --- |
| **Nr.** | **Lecture** | **Date** |
| **9** | **Head files & pre-processors** | **2022.11.4** |
| **10** | **Review of lectures** | **2022.11.11** |
| **11** | **Soul of programming: Algorithms I** | **2022.11.25** |
| **12** | **Soul of programming: Algorithms II** | **2022.12.2** |
| **13** | **R&D project** | **2022.12.9** |
| **14** | **R&D project** | **2022.12.16** |
| **15** | **R&D project** | **2022.12.23** |
| **16** | **Summary** | **2023.12.30** |

* **Pointer** is a variable that stores the address of another variable.
* We can access the **memory address** directly using the pointer.
* By changing the pointer value, the value stored at the address will be modified, typically useful for **functions** to pass values.
* Pointer can point to arrays, using **arithmetic and logical operations**

(++, --, ==, >, <) to scan the memory address.

* We can **manage the memory** using C provided functions in **stdlib.h,**

e.g. calloc(), malloc(), relloc(), free().

#### What is a pointer?

Variable Address

a stores an

integer value **int a = 10;**

# int \*b = &a;

b stores the address of

an integer variable Get the address

int \*b = &a

Variable name

int a = 10;

b is a pointer variable,

**a84ff7d0**

b0affc20

**10**

a84ff7d0

pointing to the address of a

Variable address

##### int a = 5; int \*b = &a;

* + a stores the value of 10

|  |  |  |
| --- | --- | --- |
| **Variable** | **Address** | **Content** |
| **a** | **ffc1** | **00001010** |
| **b** | **ffc2** | **ffc1** |

* + **b stores the address of a**

int \*b

b has data type int\*

**printf("%x", b);//address**

\*b has data type int

**int \*b**

**printf("%d", \*b);//value**

**int a = 5;**

**int \*b = &a;**

Use b to check the address of a

Use \*b to check the value of a

##### Pointer stores address, not value!

int a = 5; int \*b = &a;

int a = 5; int \*b = 10;

int a = 5; int \*b = &a;

\*b = 10;



int a = 5; int \*b = &a;

\*b = 10;

**int \*b = &a**

### a84ff7d0

**\*b is 5**

**int a = 10;**

### 5

a84ff7d0

##### What is a?

**a is 5**

**\*b is 10**

###### int a = 5; int \*b = &a;

Points to variable

|  |
| --- |
| **Array** |
| a[0] |
| a[1] |
| a[2] |
| a[3] |
| a[4] |
| a[5] |
| a[6] |

**int a[10]; int \*b = a;**

Points to array

Give the address of

**first element** of a to b!

###### int \*b = &a[0];

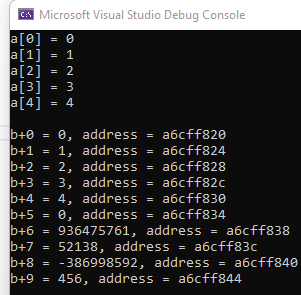
**&a[0] b**

### int a[10]; int \*b = a;

How to access the elements in array?

**下标法：a[5]**

**指针法：\*(b+5)**



**Pointer points to array**

#include<stdio.h> main()

{

int a[5] = {0, 1, 2, 3, 4};

int\* b = a;

for (int i = 0; i < 5; i++)

{

**printf("a[%d] = %d\n", i, a[i]);**

}

for (int i = 0; i < 10; i++)

{

**printf("b+%d = %d, address = %x\n",**

**i, \*(b+i), b+i);**

}

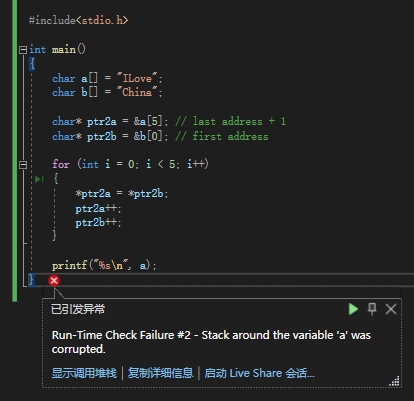
}

#include<stdio.h>

**How to concatenate 2 strings?**

## ILoveChina

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| main()  { |  | | | |
| char | a[] = "ILove"; | | | |
| char | b[] = "China"; | | | |
| char\* ptr2a = &a[5];  char\* ptr2b = &b[0]; | | // last address +  // first address | | 1 |
| for (int i = 0; i < sizeof(b);  {  \*ptr2a = \*ptr2b; ptr2a++; ptr2b++;  } | | | i++) | |

ptr2a++

**ptr2b++**

printf("%s\n", a);

}

VS 2022 may report error!

#include<stdio.h>

main()

{

**Better!**

**How to concatenate 2 strings?**

char a[100] = "ILove"; char b[] = "China";

char\* ptr2a = &a[5]; // last address + 1

char\* ptr2b = &b[0]; // first address

for (int i = 0; i < sizeof(b); i++)

{

\*ptr2a = \*ptr2b; ptr2a++; ptr2b++;

}

## ILoveChina

ptr2a++

**ptr2b++**

printf("%s\n", a);

}

#include<stdio.h>

main()

{

**How to concatenate 2 strings?**

**stop**

char a[100] = "ILove"; char b[] = "China";

char\* ptr2a = &a[5]; // last address + 1

char\* ptr2b = &b[0]; // first address

while (\*ptr2b != '\0')

{

## ILoveChina

ptr2a++

**‘\0’**

\*ptr2a = \*ptr2b; ptr2a++; ptr2b++;

}

ptr2b++

printf("%s\n", a);

}

#include<stdio.h> main()

{

**What is the length of a string?**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I** | **L** | **o** | **v** | **e** | **C** | **h** | **i** | **n** | **a** | **\0** |

char a[100] = "ILoveChina";

char\* ptr2a = &a[0];

int length = 0; while(\*ptr2a != '\0’)

{

&a[0] Stop

ptr2a++;

ptr2a++;

length++;

}

printf("Length of a is %d\n", length);

}

Length = 10

#include<stdio.h> main()

{

**What is the length of a sub-string?**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I** | **L** | **o** | **v** | **e** | **C** | **h** | **i** | **n** | **a** | **\0** |

char a[100] = "ILoveChina";

char\* ptr2a = &a[5];

int length = 0; while(\*ptr2a != '\0’)

{

ptr2a++;

length++;

}

printf("Length of a is %d\n", length);

}

&a[0] Stop

ptr2a++;

Length = 5

#include<stdio.h>

main()

{

**How to invert a string?**

char a[6] = "ABCDEF";

char\* ptr1 = &a[0]; //first address char\* ptr2 = &a[5]; //last address

int length = 0; while(ptr1 < ptr2)

{

char temp = \*ptr1;

\*ptr1 = \*ptr2;

\*ptr2 = temp;

&a[0]

ptr1++;

&a[5]

Ptr2--;

ptr1++; ptr2--;

}

Inversion is FEDCBA

printf("Inversion is %s\n", a);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **C** | **D** |

}

Pointer is typically useful for functions!!!

### Save and pass values Memory efficient (no data copy)

**Arguments Parameters**

Use pointers as **input** of function to store results!

void func(int v1, int v2, int\* sum, int\* sub, int\* mul, int\* div)

{

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | \*sum | = | v1 | + | v2; |
| \*sub | = | v1 | - | v2; |
| \*mul | = | v1 | \* | v2; |
| \*div | = | v1 | / | v2; |
| } |  |  |  |  |  |

**Input the address of 4 variables**

main()

{

int a = 10, b = 5, sum, sub, mul, div;

func(a, b, &sum, &sub, &mul, &div);

printf("sum=%d, sub=%d, mul=%d, div=%d", sum, sub, mul, div);

}

Use pointers as **output** of function to return results!

int\* func(int v1, int v2)

{

int\* ptr = (int\*) calloc(4, sizeof(int));

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ptr | + | 0 | = | v1 | + | v2; |
| ptr | + | 1 | = | v1 | - | v2; |
| ptr | + | 2 | = | v1 | \* | v2; |
| ptr | + | 3 | = | v1 | / | v2; |

return ptr;

}

**Output a pointer (array)**

main()

{

int a = 10, b = 5;

int \*ptr = func(a, b);

printf("sum=%d, sub=%d, mul=%d, div=%d", \*ptr, \*(ptr+1), \*(ptr+2), \*(ptr+3)));

}

Use pointers as **output** of function to return results!

int\* func(int v1, int v2)

{

int\* ptr = (int\*) calloc(4, sizeof(int));

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ptr[0] | = | v1 | + | v2; |
| ptr[1] | = | v1 | - | v2; |
| ptr[2] | = | v1 | \* | v2; |
| ptr[3] | = | v1 | / | v2; |

return ptr;

}

**Output a pointer (array)**

main()

{

int a = 10, b = 5;

int \*ptr = func(a, b);

printf("sum=%d, sub=%d, mul=%d, div=%d", ptr[0], ptr[1], ptr[2], ptr[3]);

}

int main(void)

{

Static array & Dynamic array

静态 动态

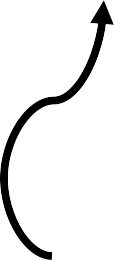
## ILove

char str1[] = "Ilove";

char\* str1\_ = (char\*)malloc(sizeof(char) \* 6);

for (int i = 0; i < strlen(str1) + 1; i++)

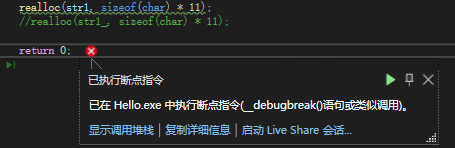
str1\_[i] = str1[i];

return 0;

}

We can convert static array to dynamic array

**We cannot use realloc() to change the size of a static array!**

int main()

{

char str1[] = "Ilove";

char\* str1\_ = (char\*)malloc(sizeof(char) \* 6);

for (int i = 0; i < strlen(str1) + 1; i++)

str1\_[i] = str1[i];

realloc(str1, sizeof(char) \* 11);

str1\_ = realloc(str1\_, sizeof(char) \* 11);

return 0;

}

The size of the static array cannot be changed !

**Variable address** as function input

**Array address** as function input

func(int \*v1, int \*v2)

{

}

main()

{

int a, b; func(&a, &b);

}

func(int v[])

{

}

func(int \*v)

{

}

main()

{

int a[10];

func(a);

}

**OR**

main()

{

int a[10];

func(a);

}

**Dynamic memory management using #include <stdlib.h>**

|  |  |
| --- | --- |
| **Function** | **Case** |
| **calloc()** | int\* a = (int\*)calloc(200, sizeof(int)); |
| **malloc()** | int\* a = (int\*)malloc(200 \* sizeof(int)); |
| **realloc()** | a = (int\*)malloc(a, 200 \* sizeof(int)); |
| **free()** | free(a); |

**Pointer to pointer**

###### int \*\*ptr; int \*ptr; int val;



**Pointer**

**Pointer**

**Variable**

**address**

**address**

Double pointer can represent matrix!

main()

{

**Single pointer**

main()

{

**Double pointer**

int r = 3, c = 4;

int\* ptr = malloc((r \* c) \* sizeof(int));

for (int i = 0; i < r \* c; i++) ptr[i] = i + 1;

for (int i = 0; i < r; i++) { for (int j = 0; j < c; j++)

printf("%d ", ptr[i \* c + j]);

printf("\n");

}

}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

int r = 3, c = 4;

int\*\* arr = (int\*\*)malloc(r \* sizeof(int\*));

for (int i = 0; i < r; i++)

arr[i] = (int\*)malloc(c \* sizeof(int));

int count = 0;

for (int i = 0; i < r; i++)

for (int j = 0; j < c; j++)

arr[i][j] = ++count;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |

### You can define your own data types in C!

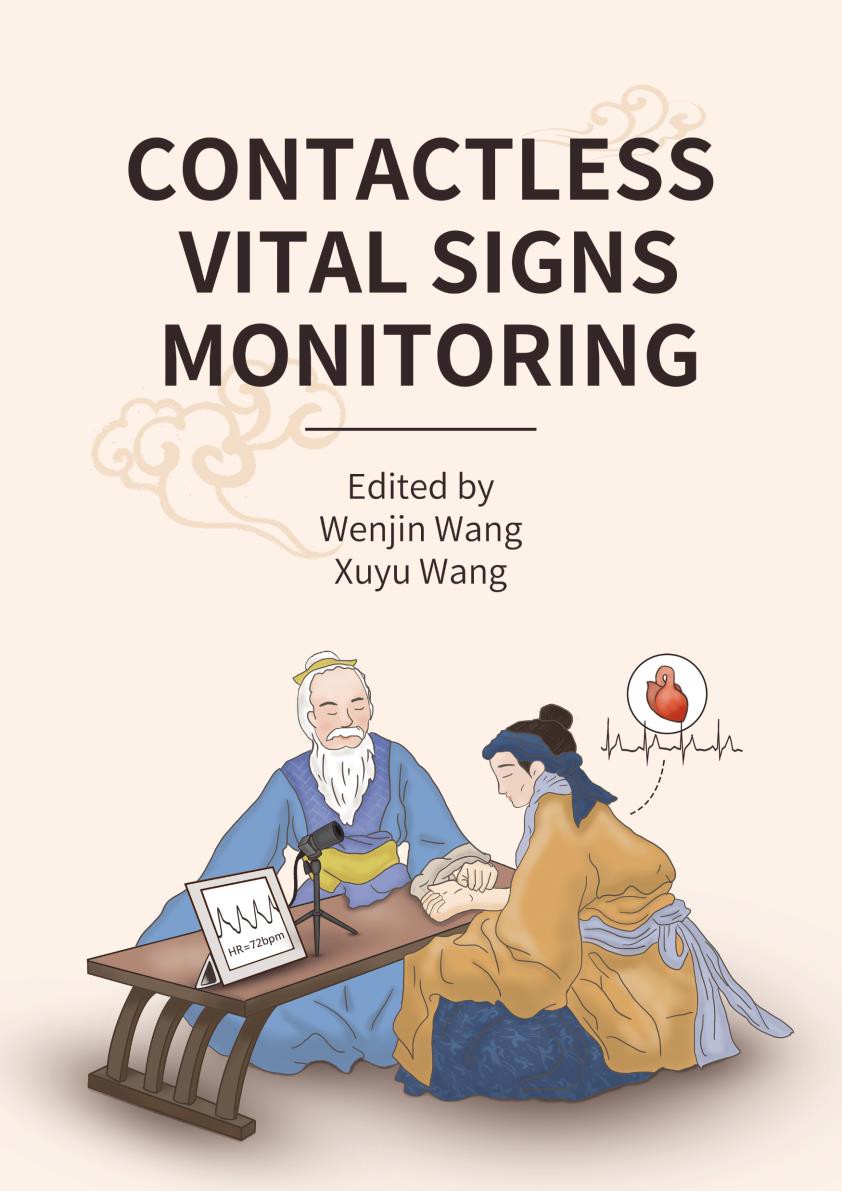
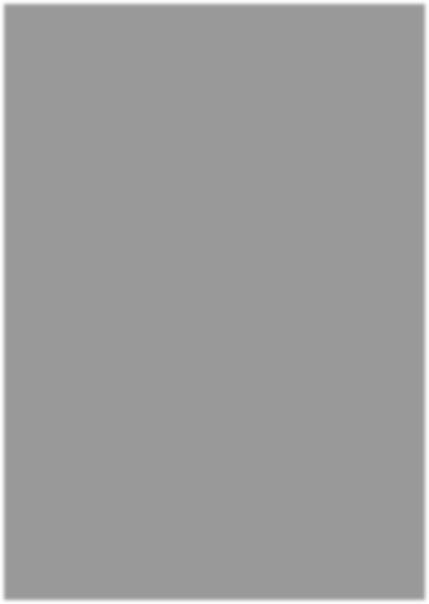
1. **Struct, union, enumerate**
2. **Typedef and #define**
3. **Struct, union, enumerate**
4. **Typedef and #define**

Student

* + Name
  + Age
  + Gender
  + ID
  + Major
  + Grade
  + Birthday

Professor

* + - Name
    - Age
    - Gender
    - Hat
    - Paper
    - Project
    - Honour



Book

* + - * Title
      * Authors
      * Publisher
      * Date
      * DOI
      * Place
      * Version

Patient

* + - * + Name
        + Age
        + Gender
        + Disease
        + Vital signs
        + Medical records
        + Symptoms

**Struct (结构体)**

**Union (共用体)**

**Enum (枚举型)**

👍👍👍 👍

Used very often Useless Used but not often

You cannot use **array** to group data with different types

# Struct (结构体)

Name

(char[])

Age

(int)

Gender

(char)

ID

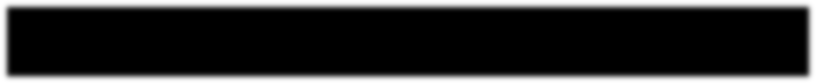
(int)

Grade

(float)

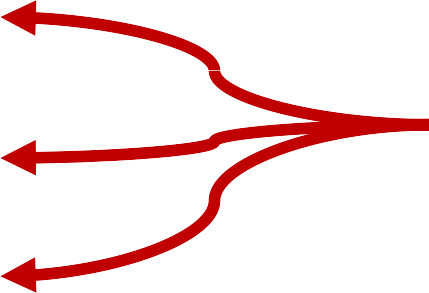
Birthday

(char[])



**Struct** defines a new data type that allows using variables with different types.

Struct name Member list



**struct name**

**{**

**type variable; type variable; type variable;**

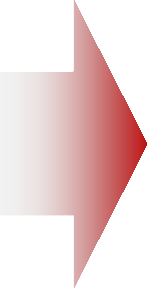
**};**

**Student Struct name**

|  |
| --- |
| **name** |
| **age** |
| **gender** |
| **ID** |
| **grade** |
| **birthday** |

**struct student**

**{**

**char name[20]; int age;**

**char gender; int ID**

**int grade;**

**char birthday[50];**

**};**

Member list

**#include<stdio.h>**

**struct student**

**{**

**char name[20];**

**int age; char gender; int ID;**

**int grade;**

**char birthday[50];**

Must define the struct data type before the main!!!

**};**

main()

{

struct student student1; strcpy(student1.name, "Jack Chen"); student1.age = 25;

student1.gender = 'M'; student1.ID = 123;

student1.grade = 80; strcpy(student1.name, "2005-October-10");

}

**Error: Expression must be a modifiable lvalue**

char name[20];

name = "Jack Chen";

#include<stdio.h>

Struct variable is defined globally!

**struct student**

**{**

**char name[20];**

**int age; char gender; int ID;**

**int grade;**

**char birthday[50];**

**} student1, student2, student3, student4;**

main()

{

strcpy(student1.name, "Jack Chen");

student1.age = 25; student1.gender = 'M'; student1.ID = 123;

student1.grade = 80; strcpy(student1.name, "2005-October-10");

}

#include<stdio.h> struct student

{

char name[20]; int age;

char gender;

int ID; int grade;

char birthday[50];

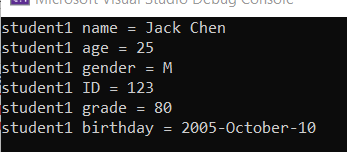
};

Initialize the struct when declaring it, must be in order!

main()

{

**struct student student1 = {"Jack Chen", 25, 'M', 123, 80, "2005-October-10"};**

printf("student1 name = %s\n", student1.name); printf("student1 age = %d\n", student1.age); printf("student1 gender = %c\n", student1.gender); printf("student1 ID = %d\n", student1.ID); printf("student1 grade = %d\n", student1.grade); printf("student1 birthday = %s\n", student1.birthday);

}

#include<stdio.h> struct student

{

char name[20];

int age; char gender; int ID;

int grade;

char birthday[50];

};

Declare and define a group of students!

main()

{

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| struct | student | student1 | = | { | "Jack Chen", | 25, | 'M', | 123, | 80, | "2005-October-10" | }; |
| struct | student | student2 | = | { | "Li Wang", | 23, | 'F', | 124, | 87, | "2004-May-9" }; |  |
| struct | student | student3 | = | { | "Steffen He", | 24, | 'M', | 125, | 92, | "2005-September-12" }; | |
| struct | student | student4 | = | { | "Tomas Huang", | 25, | 'M', | 126, | 90, | "2005-March-23" }; | |
| struct | student | student5 | = | { | "Helen Luo", | 26, | 'F', | 127, | 84, | "2005-February-15" }; | |

}

Different structs can be used in one program

**struct student**

**{**

**char name[20]; int age;**

**char gender; int ID;**

**int grade;**

**char birthday[50];**

**};**

**struct teacher**

**{**

**char name[20]; int age;**

**char gender;**

**int hat; int paper; int project int honour;**

**};**

main()

{

struct student student1 = {"Jack Chen", 25, 'M', 123, 80, "2005-October-10"};

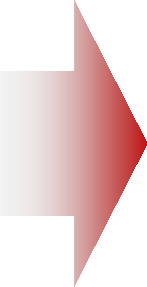
struct teacher teacher1 = {"Li Liang", 45, 'M', 1, 50, 5, 0};

}

student

|  |  |
| --- | --- |
| **name** | |
| **age** | |
| **gender** | |
| **ID** | |
| **grade** | |
| **birthday** | **Year** |
| **month** |
| **day** |

|  |
| --- |
| **name** |
| **age** |
| **gender** |
| **ID** |
| **grade** |
| **birthday** |

**birthday**

**student**

|  |
| --- |
| **year** |
| **month** |
| **day** |

**student**

**struct birthday**

**{**

**int year; int month; int day;**

|  |
| --- |
| **name** |
| **age** |
| **gender** |
| **ID** |
| **grade** |
| **birthday** |

**};**

birthday

**struct student**

**{**

**char name[20]; int age;**

|  |
| --- |
| **year** |
| **month** |
| **day** |

**char gender;**

**int ID;**

**int grade;**

**struct birthday birth;**

**};**

#include<stdio.h>

struct birthday

{

int year; int month; int day;

};

struct student

{

char name[20]; int age;

char gender; int ID;

int grade;

struct birthday birth;

};

main()

{

struct student student1; strcpy(student1.name, "Jack Chen"); student1.age = 25;

student1.gender = 'M'; student1.ID = 123;

student1.grade = 80;

**student1.birth.month = 10;**

**student1.birth.day = 10;**

**student1.birth.year = 2005;**

printf("student1 name = %s\n", student1.name); printf("student1 age = %d\n", student1.age); printf("student1 gender = %c\n", student1.gender); printf("student1 ID = %d\n", student1.ID); printf("student1 grade = %d\n", student1.grade);

**printf("student1 birthday = %d-%d-%d\n",**

**student1.birth.year, student1.birth.month, student1.birth.day);**

}

struct student

{

char name[20]; int ID;

char gender;

};

**① declare struct array and initialize it separately**

main()

{

**struct student stu[2];** strcpy(stu[0].name, "Jack"); stu[0].ID = 1;

stu[0].gender = 'M';

strcpy(stu[1].name, "Merry"); stu[1].ID = 2;

stu[1].gender = 'F';

}

struct student

{

char name[20]; int ID;

char gender;

**② declare struct array and initialize it jointly**

main()

{

**struct student stu[2] = {**

**{"Jack",1,'M'}, {"Merry",2,'F'}};**

}

};

#include<stdio.h>

**You can check the address of struct!!!**

struct student

{

char name[4];

int ID;

char gender;

};

main()

|  |  |
| --- | --- |
| **stu** | **affafb70** |
| **name** | **affafb70** |
| **ID** | **affafb74** |
| **gender** | **affafb78** |

{

struct student stu = {"Jack", 1, 'M'};

printf("Address of stu: %x\n", &stu); printf("Address of num: %x\n", &stu.name); printf("Address of ID: %x\n", &stu.ID); printf("Address of gender: %x\n", &stu.gender);

}

#include<stdio.h>

|  |  |
| --- | --- |
| **stu[0]** | **d3cff880** |
| **name** | **d3cff880** |
| **ID** | **d3cff884** |
| **gender** | **d3cff888** |
| **stu[1]** | **d3cff88c** |
| **name** | **d3cff88c** |
| **ID** | **d3cff890** |
| **gender** | **d3cff894** |

struct student

{

char name[4]; int ID;

char gender;

};

main()

{

struct student stu[2] = {{"Jack", 1, 'M'}, {"Jen", 1, 'F'}};

|  |  |  |  |
| --- | --- | --- | --- |
| for (int i = 0; i <  { | 2; | i ++) |  |
| printf("Address | of | stu: %x\n", | &stu[i]); |
| printf("Address | of | num: %x\n", | &stu[i].name); |
| printf("Address | of | ID: %x\n", &stu[i].ID); | |
| printf("Address | of | gender: %x\n", &stu[i].gender); | |

}

}

#include<stdio.h> struct student

|  |  |
| --- | --- |
| **stu[0]** | **6a6ffc20** |
| **name** | **6a6ffc20** |
| **ID** | **6a6ffc24** |
| **gender** | **6a6ffc28** |
| **stu[1]** | **6a6ffc2c** |
| **name** | **6a6ffc2c** |
| **ID** | **6a6ffc30** |
| **gender** | **6a6ffc34** |
| **stu[2]** | **6a6ffc38** |
| **name** | **6a6ffc38** |
| **ID** | **6a6ffc3c** |
| **gender** | **6a6ffc40** |

{

char name[4]; int ID;

char gender;

};

main()

{

struct student stu[3] = {{"Jack", 1, 'M'},

{"Jen", 2, 'F'}, {"Mike", 3, "M"}};

**struct student\* ptr = stu; //&stu[0]**

|  |  |  |  |
| --- | --- | --- | --- |
| printf("address | of | ptr = | %x\n", ptr); |
| printf("address | of | ptr+1 | = %x\n", ptr+1); |
| printf("address | of | ptr+2 | = %x\n", ptr+2); |

}

**ptr**

**ptr+1**

**ptr+2**

#include<stdio.h> struct student

**How to access members of struct using pointer?**

{

char name[4];

int ID;

char gender;

};

main()

{

struct student stu[3] = { {"Jack", 1, 'M'},

{"Jen", 2, 'F'}, {"Mike", 3, "M"} };

struct student\* ptr = stu; //&stu[0];

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | printf("stu | 1 | name | = | %s\n", | (\*ptr).name); |
| printf("stu | 2 | name | = | %s\n", | (\*(ptr + 1)).name); |
| printf("stu | 3 | name | = | %s\n", | (\*(ptr + 2)).name); |
|  |  |  |  |  |  |
| printf("stu | 1 | name | = | %s\n", | ptr->name); |
| printf("stu | 2 | name | = | %s\n", | (ptr + 1)->name); |
|  | printf("stu | 3 | name | = | %s\n", | (ptr + 2)->name); |

}

(\*ptr).name

**①** (\*ptr).ID

(\*ptr).gender

ptr->name

**②** ptr->ID

ptr->gender

###### Struct as input parameters for function

**Struct as output results of function**

#include<stdio.h> struct student

{

char name[5];

int ID;

char gender;

};

What is value of stu?

**input(struct student stu);**

main()

{

Jack - 1 - M

struct student stu = { "Jack", 1, 'M' };

or

**input(stu);**

printf("%s - %d - %c", stu.name, stu.ID, stu.gender);

}

**input(struct student stu)**

**{**

**strcpy(stu.name, "Lily");**

**stu.ID = 5;**

**stu.gender = 'F';**

**}**

Lily - 5 - F

#include<stdio.h> struct student

{

char name[5];

int ID;

char gender;

};

What is value of stu?

**input(struct student \*stu);**

main()

{

Jack - 1 - M

struct student stu = { "Jack", 1, 'M' };

or

**input(&stu);**

printf("%s - %d - %c", stu.name, stu.ID, stu.gender);

}

**input(struct student \*stu)**

**{**

**strcpy(stu->name, "Lily");**

**stu->ID = 5;**

**stu->gender = 'F';**

**}**

Lily - 5 - F

#include<stdio.h> struct student

{

char name[5];

int ID;

char gender;

};

Pass the array of structs to the function

**input(struct student stu[]);**

main()

{

struct student stu[2];

**input(stu);**

printf("%s - %d - %c\n", stu[0].name, stu[0].ID, stu[0].gender);

printf("%s - %d - %c", stu[1].name, stu[1].ID, stu[1].gender);

}

Lily - 5 - F Chen - 7 - M

**input(struct student stu[])**

**{**

**strcpy(stu[0].name, "Lily"); stu[0].ID = 5; stu[0].gender = 'F';**

**strcpy(stu[1].name, "Chen"); stu[1].ID = 7; stu[1].gender = 'M';**

**}**

#include<stdio.h> struct student

{

char name[5]; int ID;

char gender;

};

Return a struct from function to main

**struct student get();**

main()

{

**struct student stu = get();**

printf("%s - %d - %c\n", stu.name, stu.ID, stu.gender);

}

Lily - 5 - F

**struct student get()**

**{**

**struct student stu;**

**strcpy(stu.name, "Lily"); stu.ID = 5; stu.gender = 'F';**

**return stu;**

**}**

#include<stdio.h>

struct student

{

char name[5]; int ID;

char gender;

};

Return struct array from function to main

**struct student\* get();**

main()

{

**struct student \*stu = get();**

printf("%s - %d - %c\n", stu[0].name, stu[0].ID, stu[0].gender);

printf("%s - %d - %c\n", stu[1].name, stu[1].ID, stu[1].gender);

}

Lily - 5 - F Chen - 1 - M

**struct student\* get()**

**{**

**struct student stu[2];**

**strcpy(stu[0].name, "Lily"); stu[0].ID = 5; stu[0].gender = 'F';**

**strcpy(stu[1].name, "Chen"); stu[1].ID = 1; stu[1].gender = 'M';**

**return stu;**

**}**

### Case study: car model

#include<stdio.h>



**struct Car {**

**char brand[50]; char model[50]; int price;**

Case: if you want to buy a car after growing up?

**};**

int main()

{

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| struct | Car | car1 | = | { | "Benz", "MAYBACH", 5000000 }; |
| struct | Car | car2 | = | { | "Bentley", "Flying Spur", |
| 4000000 }; |  |  |  |  |  |
| struct | Car | car3 | = | { | "Maserati", "Quattroporte", |
| 3000000 }; |  |  |  |  |  |

printf("%s %s %d\n", car1.brand, car1.model, car1.price);

printf("%s %s %d\n", car2.brand, car2.model,

car2.price);

printf("%s %s %d\n", car3.brand, car3.model, car3.price);

}

### Case study: calculate GPA

Case: calculate your GPA!

#include<stdio.h>

struct course

{

char name[20]; int score; float credit;

};

float cal\_GPA(struct course course[], int num)

{

float GPA = 0;

float weight = 0;

for (int i = 0; i < num; i++)

{

GPA += course[i].credit \* course[i].score; weight += course[i].credit;

}

return GPA / weight;

}

|  |  |  |  |
| --- | --- | --- | --- |
| **科目** | **C程序基础** | **高等数学** | **线性代数** |
| 得分 | 98 | 85 | 90 |
| 学分 | 4 | 5 | 3.5 |

**GPA = (98 × 4 + 85 × 5 + 90 × 3.5) / (4 + 5 + 3.5) = 90.56**



main()

{

struct course course\_[3];

course\_[0] = { "C程序设计基础",98,4 };

course\_[1] = { "高等数学",85,5 };

course\_[2] = { "线性代数",90,3.5 }; printf("Your GPA is %f", cal\_GPA(course\_, 3));

}

### Case study: transmit packet

Case: verify if the transmission is correct?

#include<stdio.h>

struct point

{

float x; float y;

float distance;

float verify;

};

cal\_verify(struct point \*in)

{

in->verify = in->x + in->y + in->distance;

}

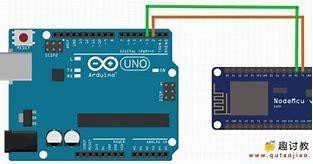
main()

{

struct point p = { 2.5, 2.5, 6, 0 };

cal\_verify(&p); printf("%f", p.verify);

}

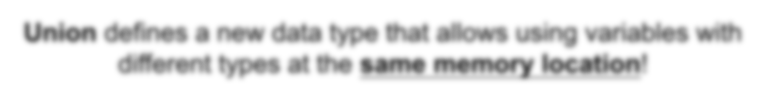
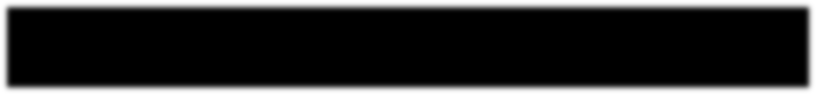


Transmit 3 values (a, b, a+b), check if received

values are correct

### When to use struct?

When you want to group different types of data in a single unit!



**Union** defines a new data type that allows using variables with different types at the **same memory location**!

**union [union tag]**

**{**

**type variable;**

**type variable;**

**…**

**};**

**Union (共用体)**

**struct student**

**{**

**char name[20]; int ID;**

**int grade;**

**};**

**union student**

**{**

**char name[20]; int ID;**

**int grade;**

**};**

**sizeof(student) = 20**

**20 bytes = max(20, 4, 4)**

**Store at the same memory location!!!**

**#include <stdio.h> #include <stdio.h>**

**union Data**

**{**

**int i; float f;**

**char str[20];**

**};**

data.i : 1917853763

data.f : 4122360580327794860452

759994368.000000

data.str : C Programming

**struct Data**

**{**

**int i; float f;**

**char str[20];**

**};**

data.i : 10

data.f : 220.500000

data.str : C

Programming

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);

}

main( )

{

struct 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);

}

**#include <stdio.h>**

main( )

**union Data**

**{**

**int i; float f;**

**char str[20];**

**};**

{

data.i: 10

data.f: 220.500000

data.str: C Programming

union Data data; data.i = 10;

printf( "data.i : %d\n", data.i);

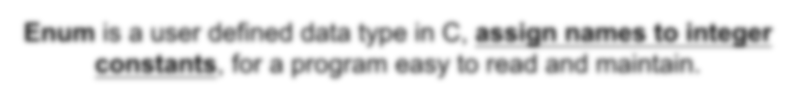
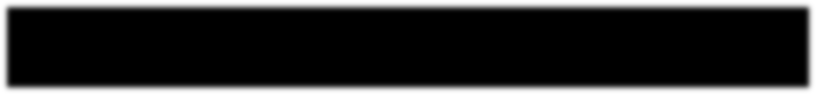
data.f = 220.5;

printf( "data.f : %f\n", data.f); strcpy(data.str, "C Programming"); printf( "data.str : %s\n", data.str);

}

**Do NOT use union,**

**use struct as much as you can!!!**



**Enum** is a user defined data type in C, **assign names to integer constants**, for a program easy to read and maintain.

**enum [union tag]**

**{**

**variable;**

**variable;**

**…**

**};**

**All integers by default!**

**Enum (枚举型)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| enum | week | { | Mon, | Tue, | Wed, | Thu, | Fri, | Sat, | Sun | }; |
|  |  |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| enum | week | { | Mon=1, | Tue, | Wed, | Thu, | Fri, | Sat, | Sun | }; |
|  |  |  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |  |

enum week day;

day = Mon;

Enum assigns names to integer constants

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #include<stdio.h> |  |  | #include<stdio.h> |  |
| **enum week { Mon, Tue, Wed, Sat, Sun };** | **Thur,** | **Fri,** | **enum week { Mon, Tue, Wed, Sat, Sun };** | **Thur, Fri,** |
| main()  {  **enum week day;**  for (day = Mon; day <=  { | Sun; | day++) | main()  {  for (int i = Mon; i <=  {  printf("%d\n", i);  } | Sun; i++) |

printf("%d\n", day); }

}

}

Enum assigns names to integer constants

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| #include<stdio.h> | | | |  | #include<stdio.h> | |  |
| **enum Year { Jan, Feb, Mar, Apr, May,**  **June, July, Aug, Sept, Oct, Nov, Dec** | | | | **};** | **enum Year { Jan,**  **June, July, Aug,** | | **Feb, Mar, Apr, May,**  **Sept, Oct, Nov, Dec };** |
| main()  {  enum Year year; | | | |  | main()  {  for (int i =  { | | Jan; i <= Dec; i++) |
| for (year  { | = Jan; | year <= Dec; | year++) | | } | printf("%d\n", i); | |

printf("%d\n", year); }

}

}

### Case study: check weekday

#include <stdio.h>

main() {

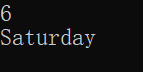
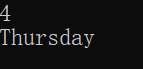
enum week { Mon = 1, Tues, Wed,

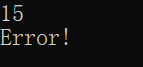
Thurs, Fri, Sat, Sun } day; scanf\_s("%d", &day); switch (day) {

case Mon: puts("Monday"); break; case Tues: puts("Tuesday"); break; case Wed: puts("Wednesday"); break; case Thurs: puts("Thursday"); break; case Fri: puts("Friday"); break; case Sat: puts("Saturday"); break; case Sun: puts("Sunday"); break; default: puts("Error!");

}

Case: input a number, check the weekday



}

### Case study: check season

Case: is it spring now?

#include <stdio.h>

enum Season{spring, summer, fall, winter};

main()

{

enum Season now = spring;

if (now < summer)

{

printf("It's spring now");

}

}



**When to use enum?**

#### When you want to assign a sequential of names with integers

👍 **Jan, Feb, Mar, Apr, May, June, July, Aug, Sept, Oct, Nov, Dec**

👍 **Mon, Tue, Wed, Thu, Fri, Sat, Sun Jack, Lily, Tom, John, Wim, Kevin, Henk**

**Struct (结构体)**

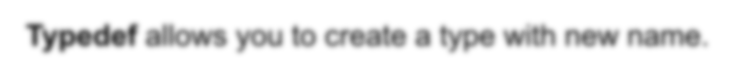
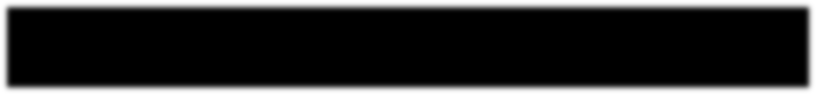
**Union (共用体)**

**Enum (枚举型)**

👍👍👍 👍

Used very often Very useless Used but not often

1. **Struct, union, enumerate**
2. **Typedef and #define**



**Typedef** allows you to create a type with new name.

**Typedef type name;**

**typedef** unsigned char **BYTE**; **typedef** struct Books

{

char title[50]; char author[50]; int book\_id;

} **Book**;

Variable

**unsigned char** var1; **unsigned char** var2; **unsigned char** var3;

Variable with typedef

**typedef** unsigned char **BYTE**; **BYTE** var1;

**BYTE** var2;

**BYTE** var3;

**A new type**

Struct

struct Books

{

char title[50]; char author[50]; char subject[100]; int book\_id;

};

Struct with typedef

**typedef** struct Books

{

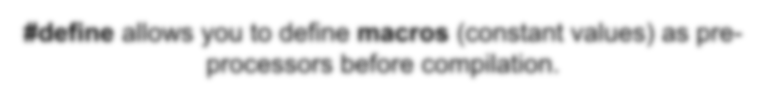
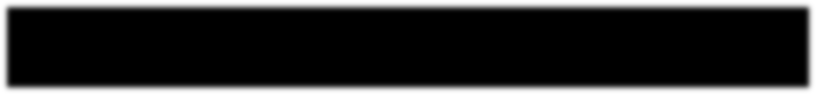
char title[50]; char author[50]; char subject[100]; int book\_id;

} **Book**;

**struct Books** book1; **struct Books** book2; **struct Books** book3;

**Book** book1; **Book** book2; **Book** book3;

**A new type**



**#define name value**

**#define** allows you to define **macros** (constant values) as pre- processors before compilation.

Macro (宏) definitions must be constant, there is no symbols like = and ;

**#define sets macros for numbers, strings or expressions**

**numbers**

**strings**

**Expr.**

#define ONE 1

#define PI 3.14 #define CHAR 'a' #define NAME "SUSTech"

#define MIN(a, b) (a < b ? a : b)

#define SUM(a, b, c) (a + b + c)

Use const global variable

#include<stdio.h>

**const int ONE = 1; const float PI = 3.14; const char CHAR = 'a';**

**const char NAME[] = "SUSTech";**

Use #define macro

#include<stdio.h>

**#define ONE 1**

**#define PI 3.14 #define CHAR 'a' #define NAME "SUSTech"**

|  |  |  |  |
| --- | --- | --- | --- |
| main() |  | main() |  |
| { |  | { |
| printf("%d\n", | ONE); | printf("%d\n", | ONE); |
| printf("%f\n", | PI); | printf("%f\n", | PI); |
| printf("%c\n", | CHAR); | printf("%c\n", | CHAR); |
| printf("%s\n", | NAME); | printf("%s\n", | NAME); |
| } |  | } |  |

#include<stdio.h>

**#define MIN(a, b) (a < b ? a : b) #define SUM(a, b, c) (a + b + c) #define POW(x) (x \* x)**

main()

{

**Use #define for macro expressions**

|  |  |  |
| --- | --- | --- |
| printf("%d\n", | MIN(10, | 100)); |
| printf("%d\n", | SUM(10, | 20, 30)); |
| printf("%f\n", | SUM(2.3, | 0.8, -3.5)); |
| printf(“%d\n”, | POW(5)); |  |

}

**Write code**

typedef

**Compile Link**



**.c source code**



**.o objective code**



**.exe program**

**Preprocess**

#define #include

**C source code Preprocessed code**

#include<stdio.h>

#define SQUARE(X) X\*X #define PR(x) printf("The result is %d.\n",x)

int main()

{

int z = 5; PR(SQUARE(z));

PR(SQUARE(z + 2)); PR(100 / SQUARE(2));

return 0;

}

int main()

{

int z = 5;

printf("The result is %d.\n",

**z\*z**);

printf("The result is %d.\n", (**z + 2)\*(z + 2)**);

printf("The result is

%d.\n",**100/(2\*2)**);

return 0;

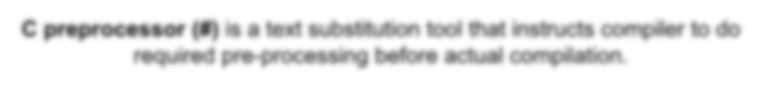
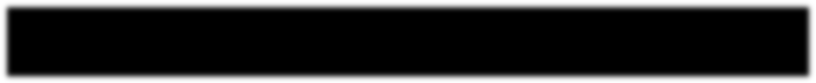
}

##### typedef

* Processed by **compiler**, actual definition of a new type
* Give symbolic names to types
* With scope rules. If defined inside function, only visible to the function

##### #define

* Processed by **preprocessor**, copy-paste the definition in place
* Define alias for values (#define ONE 1)
* No scope rules, it replaces all occurrences, visual everywhere



**C preprocessor (#)** is a text substitution tool that instructs compiler to do required pre-processing before actual compilation.

#define Substitutes a preprocessor macro

#include Inserts a particular header from another file

#undef Undefines a preprocessor macro

#ifdef Returns true if this macro is defined #ifndef Returns true if this macro is not defined

#if Tests if a compile time condition is true

#else The alternative for #if

#elif #else an #if in one statement #endif Ends preprocessor conditional

#error Prints error message on stderr

#pragma Issues special commands to the compiler, using a standardized method

### Case study: area of circle

Case: calculate the area of a circle.

# include <stdio.h> # define PI 3.14159

main()

{

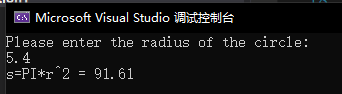
float r, s;

printf("Please enter the radius of the circle:\n");

scanf\_s("%f", &r); s = PI \* r \* r;

printf("s=PI\*r^2 = %.2f\n", s);

}



### Case study: basic statistics

Case: calculate the maximum, minimum, mean of two values

#include<stdio.h>

#define MAX(X,Y) (X>Y ? X:Y)

#define MIN(X,Y) (X<Y ? X:Y)

#define MEAN(X,Y) (X+Y/2)

int main(void)

{

float x, y;

printf("Enter two numbers:\n"); scanf\_s("%f %f", &x, &y); printf("The mean is %f.\n", MEAN(x,

y));

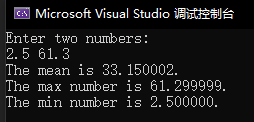
printf("The max number is %f.\n",

MAX(x, y));

printf("The min number is %f.\n", MIN(x, y));

return 0;

}



##### Case study: point to line distance

Case: calculate the distance from point to line

#include<stdio.h>

#include<math.h>

#define PRINT(x) printf("The result is %f.\n",x)

#define ABS(X) (X < 0 ? -X:X)

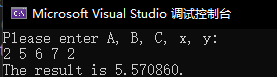
printf("Please enter A, B, C, x, y:\n");

scanf\_s("%d %d %d %d %d", &A, &B, &C, &x, &y); float dis = DST(A, B, C, x, y);

PRINT(dis);

return 0;

}



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| float DST(int A, int B,  { | | | int | C, int x, int y) |
| float distance = (A | | | \* x | + B \* y + C) / sqrt(A \* |
| A + B \* B); | | |  |  |
| } | return | ABS(distance); | | |
| int | main()  int A, | {  B, C, x, y; | | |

1. **Struct, union, enumerate**
2. **Typedef and #define**

* **Struct** can be used to define a new data type for grouping data with different types.
* Struct is very useful and has been commonly used. It can be used with

**arrays, pointers, and functions**.

* **Union** is not useful (only need know). **Enum** can be used to assign a sequence of names with integers.
* **Typedef** can define a new type of data by combining existing ones (short int, struct), **#define** can define marcos for pre-processing.
* Time to define your own data types!

1. There are 10 students, you know their names, student numbers and temperatures.

Who has a fever? If anyone has a fever, print his/her name and student number.

1. The fever is defined as the temperature that is higher than 37.3℃.
2. Hint: you can use an array of structs to store name, student number and temperature of 10 students.
3. Below is a form that contains the information of the 10 students.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student number** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| Name | Jack | Tim | Steve | Mara | Paula | Emma | David | Tyler | Noah | Daniel |
| Temperature | 36.5 | 37.6 | 35.9 | 36.9 | 38.1 | 35.4 | 36.1 | 37.9 | 36.8 | 37.1 |

1. Below form contains the temperature of this week. Write a function that uses a struct that includes date, day (e.g. Monday) and temperature as argument, and prints the date, day if the temperature is higher than 30℃.

a) Hint: you can use enum to enumerate the day (Monday, Tuesday ……) and use

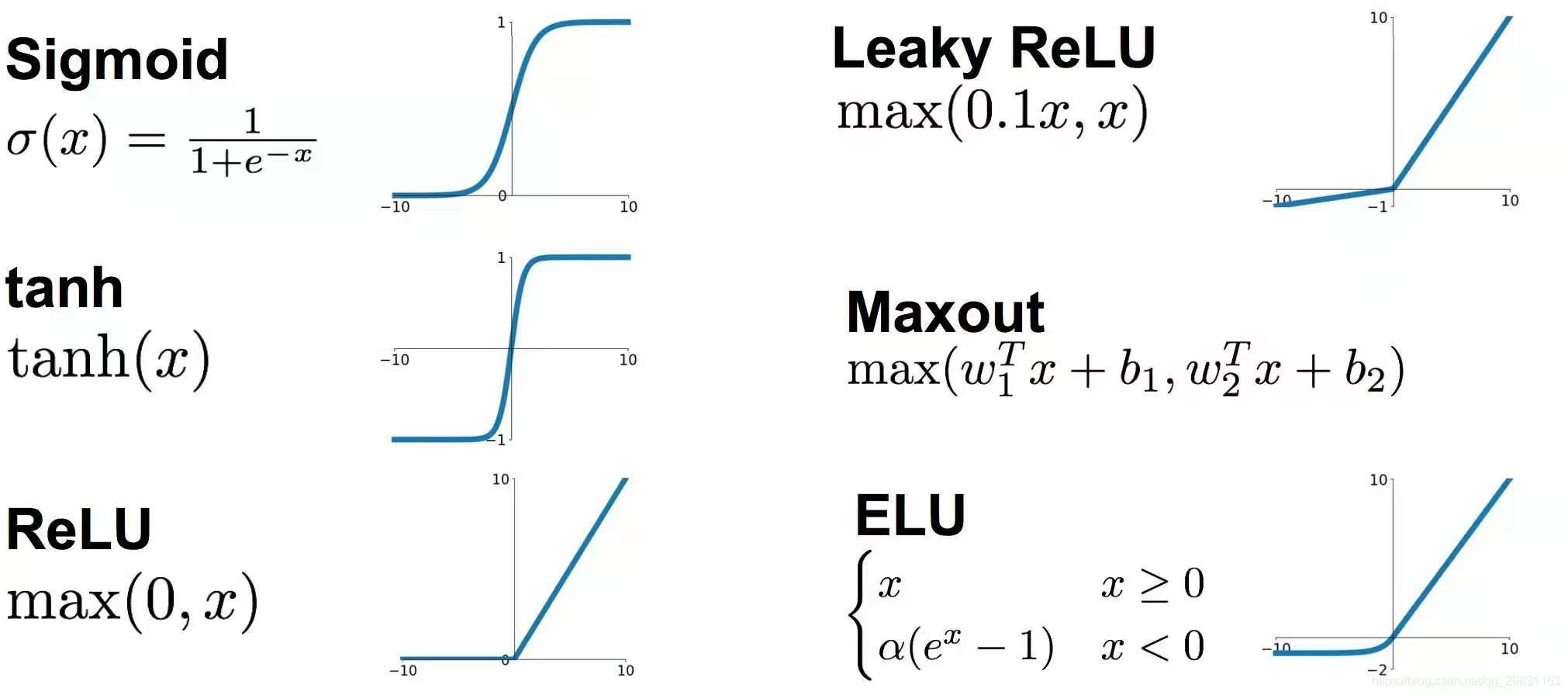
struct to store the date, day and temperature.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Date** | **10.17** | **10.18** | **10.19** | **10.20** | **10.21** | **10.22** | **10.23** |
| Day | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |
| Temperature | 28 | 27 | 27 | 28 | 30 | 32 | 31 |

1. You already know how to use function in C to implement following activation

functions, use #define (macros) to implement the activation functions.

a) Test input : -0.5, 9



tanh 𝑥

=

𝑒𝑥 − 𝑒−𝑥

𝑒𝑥 + 𝑒−𝑥

1. Below are X and Y coordinates of 20 points. There is another point which is called

P. Calculate the distances between P and each points and print the distance and coordinate of each point.

1. Hint: you can use an array of structs to store the distance and coordinate of each point.
2. The coordinate of the points are as follows, you can copy and paste them directly into your code.

struct point

{

int x; int y;

};

struct point pts[20] =

{ {49,95},{58,93},{78,6},{20,80},{86,97},{65,60},{92,23},{34,43},{65,69},{13,80},{22,56},{72

,41},{67,22},{81,14},{41,84},{61,42},{78,93},{2,30},{73,33},{45,2}};

1. Here is a famous lyrics. Count the number of occurrences of each word and print the words with its number of occurrences.
2. Hint 1: each word is a string (1-D array of char), so you can use a 2-D array of

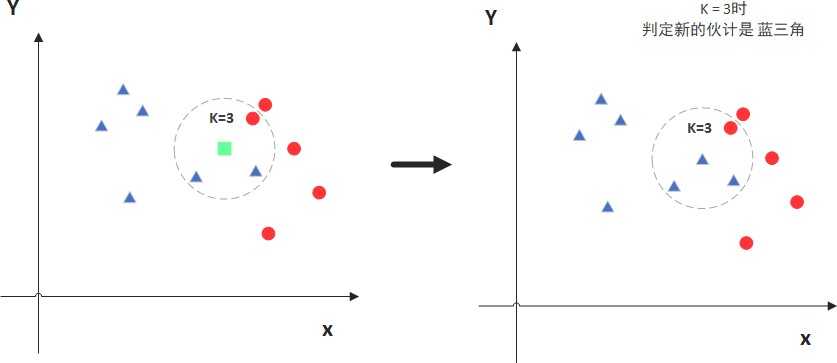
char to store the text .

1. Hint 2: you can use an array of structs to store the number of occurrences of each word and the word.
2. Hint 3: you can directly copy below array in C for processing.

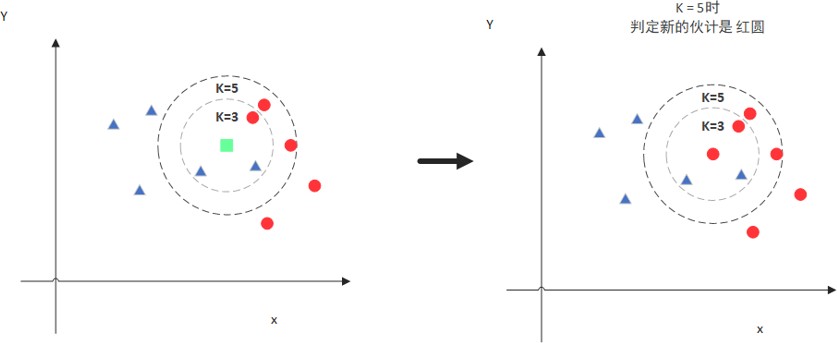
const char text[40][20] = {

"Every","night","in","my","dreams", "I","see","you", "I","feel","you","that","is","how","I","know","you","go","on","far","across","the","dist

ance","and","spaces","between","us","you","have","come","to","show","you","go","on","nea r","far","wherever","you","are"};

1. (**bonus**) The full name of KNN is K Nearest Neighbors. The principle of KNN is that giving a new sample, it determines which category the sample belongs to by checking the K nearest points in the training data. K is called hyperparameter, defined by users. The green dot in the graph is the sample we want to predict, assuming K = 3. The KNN algorithm will find the three points closest to it (in the dash circle) and check what is the category of majority of K nearest points. In this example, there are more blue triangles, so the new sample (green point) is classified as the blue triangle.

However, when K=5, the decision is different. According to the rule, the new sample (green point) is classified as the red circle. From this example, we can see that the definition of K is important.



Here is 20 points in total, 10 points belong to category A, 10 points belong to category B. Given a new point P, use KNN to determine which category P belongs to. a) Set K = 3, P = (45, 60).

b) The coordinates and categories of 20 points are as follows, you can copy and paste them directly into your code.

typedef enum category { A, B } Enum;

typedef struct point

{

int x; int y;

float dist; Enum category;

}Point;

Point points[20] = { 49,95,0,A,58,93,0,A,78,6,0,A,20,80,0,A,86,97,0,A,65,60,0,A,92,23,0,A,34,43,0,A,65,69,0,A,13,80,0,A,22

,56,0,B,72,41,0,B,67,22,0,B,81,14,0,B,41,84,0,B,61,42,0,B,78,93,0,B,2,30,0,B,73,33,0,B,45,2,0,B };