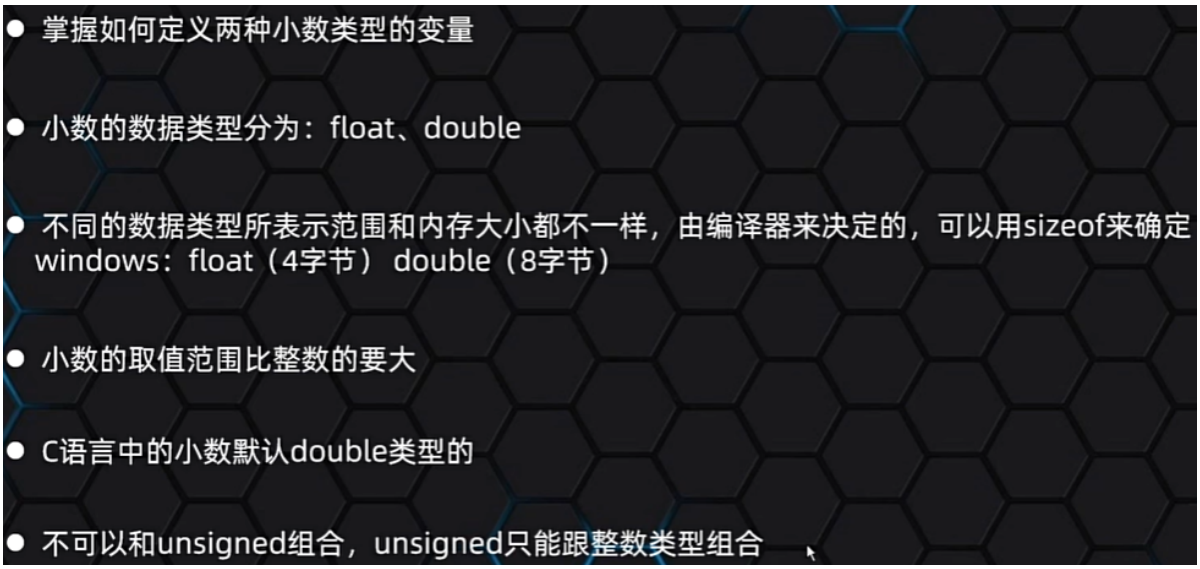
# 整型

```cpp
// practice.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <iostream>

int main()
{
    /*  short短整型2个字节(-32768 ~ 32767)
        int整型4个字节(-2147483648 ~ 2147483647)
        long长整型4个字节(-2147483648 ~ 2147483647)
        long long (c99)
        */
    short a = 10;
    int b = 100;
    long c = 1000L;
    long long d = 10000LL;
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("c = %ld\n", c);
    printf("d = %lld\n", d);
    printf("short类型长度为%zu个字节\n", sizeof(short));
    printf("int类型长度为%zu个字节\n", sizeof(int));
    printf("long类型长度为%zu个字节\n", sizeof(long));
    printf("long long类型长度为%zu个字节\n", sizeof(long long));

    //unsigned short （0~65535）
    unsigned short a1 = 999;
    unsigned int b1 = 999;
    unsigned long c1 = 999L;
    unsigned long long d1 = 999LL;
    printf("a1 = %u\n", a1);
    printf("b1 = %u\n", b1);
    printf("c1 = %lu\n", c1);
    printf("d1 = %llu\n", d1);

    return 0;
}
```


## 浮点型

```c
# include <stdio.h>

int main() {
    // 1. 定义float、double、long、double数据类型的变量
    // float 单精度小数（精确度小数点后6位）windows占4个字节(38位)
    // double 双精度小数（精确度小数点后15位）windows占8个字节（308位）
    // long double 高精度小数（精确度小数点后18~19位）windows占8个字节
    float a = 3.14F;
    printf("%f\n", a);
    printf("%.2f\n", a);//小数点后保留2位

    double b = 1.78;
    printf("%lf\n", b);
    printf("%.2lf\n", b);

    long double c = 3.1415926L;
    printf("%.7lf\n", c);
    printf("%.2lf\n", c);

    printf("%zu\n", sizeof(a));
    printf("%zu\n", sizeof(b));
```

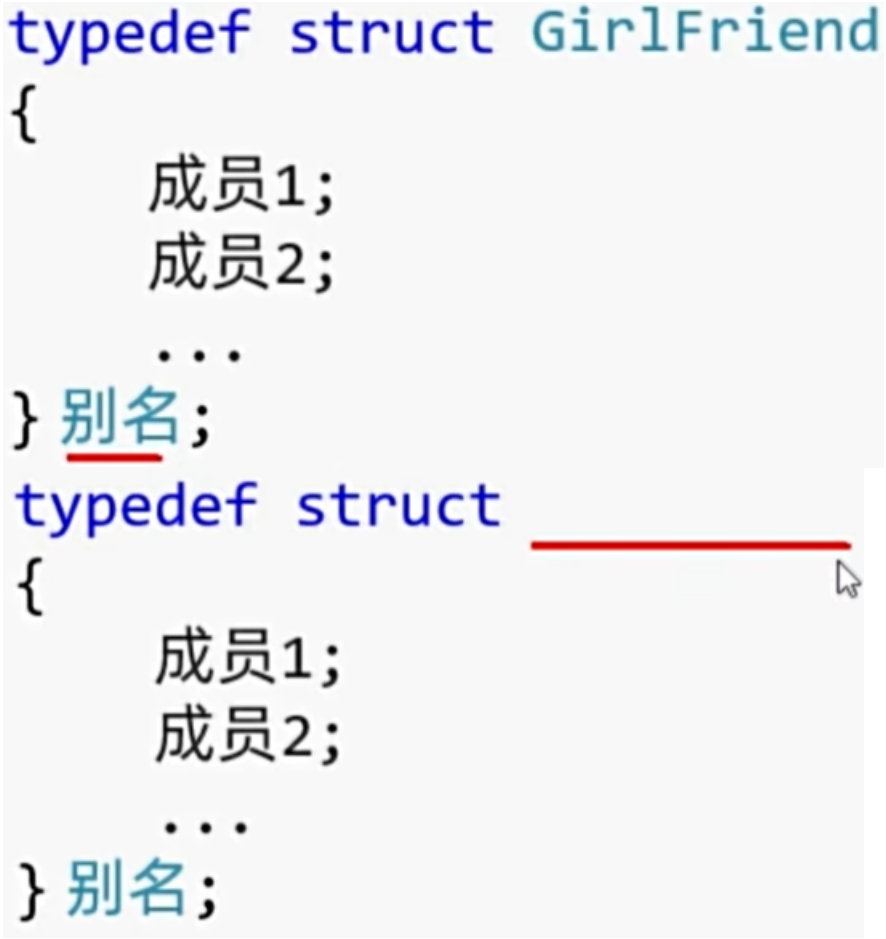
```
    printf("%zu\n", sizeof(c));
    return 0;
}
```

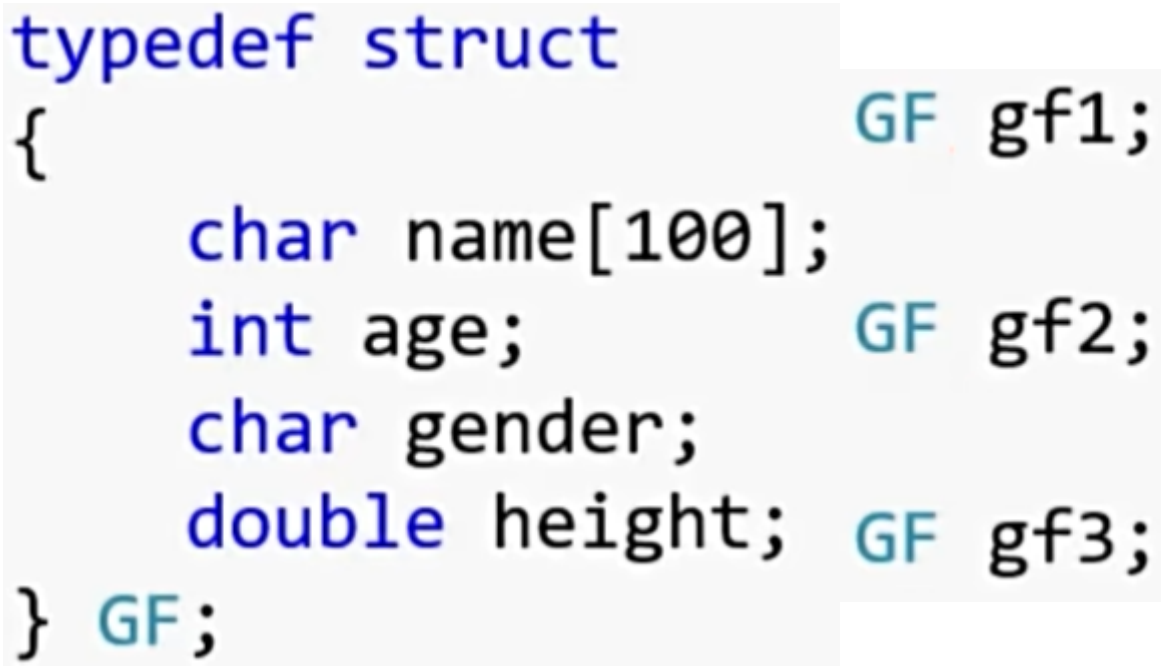
# 结构体

```
#include <string.h>
#include <stdio.h>

/*
    结构体：
        自定义的数据类型
        由很多的数据组合成的一个整体
        每一个数据，都是结构体的成员

    书写的位置：
        函数的里面：局部位置，只能在本函数中使用
        函数的外面：全局位置，在所有的函数中都可以使用

*/

struct GirlFriend
{
    char name[100];
    int age;
    char gender;
    double height;
};

int main() {
    // 使用结构体
    // 定义一个女朋友类型的变量
    struct GirlFriend gf1;
    strcpy(gf1.name, "张盈");
    gf1.age = 23;
    gf1.gender = 'F';
    gf1.height = 1.63;

    // 输出打印
    printf("My girlfriend's name is: %s\n", gf1.name);
    printf("My girlfriend's age is: %d\n", gf1.age);
    printf("My girlfriend's gender is: %c\n", gf1.gender);
    printf("My girlfriend's height is: %.2lf\n", gf1.height);
    return 0;
}
```

```
# include <stdio.h>
# include <string.h>

struct Student {
    char name[100];
    int age;
};
```

```c
int main() {
    //1.定义两个学生，同时进行赋值
    struct Student stuA = { "张盈", 24 };
    struct Student stuB = { "汪思斌", 25 };

    //2.把三个学生放入到数组当中
    struct Student StuArr[2] = { stuA, stuB };

    //3. 遍历数组得到每一个元素
    for (int i = 0; i < 2; i++) {
        struct Student temp = StuArr[i];
        printf("学生的信息为：姓名%s，年龄%d\n", temp.name, temp.age);
    }
}
```






```c
#include <string.h>
#include <stdio.h>

typedef struct {
    char name[100];
    int attack;
    int defense;
    int blood;
} M;

int main() {
    /*
        定义一个结构体表示游戏人物
            属性有：姓名、攻击力、防御力、血量
            要求：把三个游戏人物放入到数组当中，并遍历数组
    */

    //1.定义三个奥特曼
    M taro = { "泰罗", 100, 90, 500 };
    M rem = { "雷欧", 90, 80, 450 };
    M eddie = { "艾迪", 120, 70, 600 };

    //2.定义数组
    M arr[3] = { taro, rem, eddie };

    //3.遍历数组
    for (int i = 0; i < 3; i++) {
        M temp = arr[i];
        printf("奥特曼的名字为%s,攻击力为%d,防御力为%d,血量是%d\n", temp.name,
temp.attack, temp.defense, temp.blood);
    }
    return 0;
}
```

利用函数修改main函数中的结构体的变量

```c
# include <stdio.h>
# include <string.h>

typedef struct {
    char name[100];
    int age;

}S;

void method(S ss); //这个函数用到了结构体，因此要写在结构体下面。
void method2(S* p);

int main() {
    S stuA = { "张盈", 18 };
    method2(&stuA); // 函数中传的参数如果是指针，则需要加&取地址

    printf("接收到method函数修改后的学生名字为%s，年龄为%d\n", stuA.name, stuA.age);
    return 0;
}
/*
    细节：
        如果函数中写的是结构体类型的变量，相当于是定义了一个新的变量
        此时是把main函数中stu中的数据，传递给了method函数，并把stu中的数据赋值给了新的变量st
        我们在函数中，仅仅是修改了变量st中的值，对main函数中stu的值，是没有进行修改的

*/
void method(S ss) {
    printf("接收到main函数的学生名字为%s，年龄为%d\n", ss.name, ss.age);
    // 修改
    printf("输入修改后的名字：");
    scanf("%s", ss.name); // name是一个数组，此处不再需要&，因为ss.name就表示地址
    printf("输入修改后的年龄：");
    scanf("%d", &(ss.age)); // int是一个整型，不是数组，必须用&
    printf("method函数修改后的学生名字为%s，年龄为%d\n", ss.name, ss.age);
}

//如果要在函数中修改stu的值，此时就不要再定义新的变量了
//直接接收stu的内存地址，通过内存地址就可以修改stu中的数据了
//指针p里面记录的是main函数中stu的内存地址(stu 学生)

void method2(S* p) {
    printf("接收到main函数的学生名字为%s，年龄为%d\n", (*p).name, (*p).age);
    // 修改
    printf("输入修改后的名字：");
    scanf("%s", (*p).name); // name是一个数组，此处不再需要&，因为ss.name就表示地址
    printf("输入修改后的年龄：");
    scanf("%d", &((*p).age)); // int是一个整型，不是数组，必须用&
    printf("method函数修改后的学生名字为%s，年龄为%d\n", (*p).name, (*p).age);
}
```

# 数组

数组求和

```c
# include <stdio.h>
```

```c
# include <stdlib.h>
# include <time.h>


int main() {
    int arr[10] = { 0 };

    int len = sizeof(arr) / sizeof(arr[0]);

    // 生成10个1~100之间的随机数
    // 设置种子
    srand(time(NULL));

    for (int i = 0; i < len; i++) {
        // 生成随机数
        int num = rand() % 100 + 1;
        arr[i] = num;
    }
    for (int i = 0; i < len; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
    }
    int sum = 0;
    for (int i = 0; i < len; i++) {
        sum = sum + arr[i];
    }
    printf("数组元素之和为：%d\n", sum);

    return 0;
}
```

数组求和（升级版）

```c
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
int contentSameVal(int arr[], int len, int val);
int main() {
    int arr[10] = { 0 };
    srand(time(NULL));
    int len = sizeof(arr) / sizeof(arr[0]);
    for (int i = 0; i < len; )
    {
        int num = rand() % 10 + 1;
        int flag = contentSameVal(arr, len, num);
        if (!flag) {
            arr[i] = num;
            i++;
        }
    }
    for (int i = 0; i < len; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
    }
    return 0;
}
```

```
int contentSameVal(int arr[], int len, int val) {
    for (int i = 0; i < len; i++) {
        if (arr[i] == val)
            return 1;
    }
    return 0;
}
```

基本查找

二分查找

```
# include <stdio.h>
int binarySearch(int arr[], int len, int num);

int main() {
    int arr[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int len = sizeof(arr) / sizeof(arr[0]);
    int num = 2;

    int index = binarySearch(arr, len, num);

    printf("%d\n", index);
    return 0;
}

int binarySearch(int arr[], int len, int num) {

    int min = 0;
    int max = len - 1;

    while (min <= max) {
        int mid = (max + min) / 2;
        if (num > arr[mid]) {
            min = mid + 1;
        }
        else if (num < arr[mid]) {
            max = mid - 1;
        }
        else {
            return mid;
        }
    }

    return -1;
}
```

插值查找

分块查找和哈希查找