数组、函数、结构体、枚举类型

数组++

初步理解数组

之前我们介绍过变量，那么大家有没有想过，在我们需要大量类似功能的变量（比如对50位同学的语文成绩进行计算）时，有没有一种办法能够一次性把这些变量创建好呢?

那么数组就很适合这种场景。 数组可以理解为一种带有下标的变量，比如“a[50]”，这就表示我们创建了一个有50个元素的数组（可以理解为多个变量）。

我们可以通过改变下标（就是 [ ] 内的数字）来访问不同的变量，比如“a[0]”表示数组中首个元素（没错我们程序猿是 从0开始数数的），“a[49]”表示数组中最后一个元素。

补充点

在变量创建时，比如：

```
1 int a[50];
```

这时的"50"表示这个数组有50个元素，但是对于这个数组的下标却是"0"到"49"这50个，也就是说数组的下标是 从0开始计算的，这一点要额外注意（所以为了避免出现奇奇怪怪的bug，假如我们需要100个元素的数组，往往会多创建10个，反正目前来看我们的代码瓶颈不存在存储方面）。 数组在使用的时候比较灵活，我们可以通过变量或常量来访问数组下标，比如：

```
1 int a[50]={0};
2 int b=a[0];
3 int c=a[b];
```

以上使用方法都是合法的。

我们还可以把数组的范围扩大一些，然后从第 1 位开始用不是从第 0 位开始用，这样比较符合直觉。

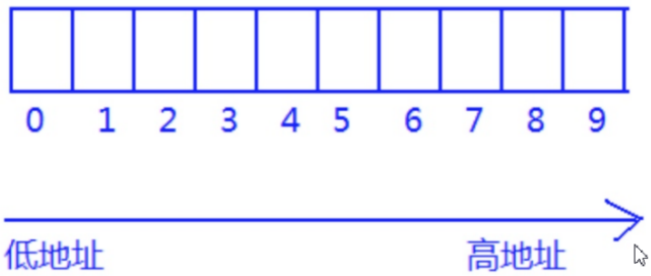
数组的基本概念

- 数组是相同类型数据的有序集合
- 这些相同类型的数据按照一定的先后次序排列组合而成

- 其中每一个数据称作一个数组元素，每个数组元素可以通过一个下标来访问它们
- 数组的下标从0开始到(n-1)

数组内部储存原理

在我们想存储一连串数据之前，需要先申请一块连续的内存空间，这个空间的大小在程序的运行过程中不可改变。



一维数组

定义一维数组

数据类型 数组名[常量表达式]

例如：

```
1 int a[ 4 ] ;      //创建一个有四个整型变量的数组
```

此时a数组中的元素：

a[0]	a[1]	a[2]	a[3]

应用一维数组元素

上面我们定义了一个有四个元素的数组，接下来我们给每一个数组元素赋值：

```
1 a[0]=1;
2 a[1]=2;
3 a[2]=3;
4 a[3]=4;
```

简单粗暴的赋值方式，但还可以使用下面一种方法赋值：

```
for(i=0;i<3;i++)
    a[i]=i+1;
```

一维数组的初始化

在定义数组时对全部元素赋予初值：

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

或者

```
int a[] = {1,2,3,4};//当[ ]内的值与{ }内的元素个数相同时，[ ]内可以省略
```

初始化之后有a[0]=1, a[1]=2,a[2]=3,a[3]=4

特殊的：

```
int a[4] = {1};
```

会将a[0]赋值为1，其他都初始化为0

程序举例

```
#include <stdio.h>
#define SIZE 1024

int main()
{
    int n;
    printf("请输入字符个数:");
    scanf("%d", &n);
    char a[n + 1]; //此行只能在gcc编译器下使用
    //此数组范围为a[0]~a[n-1]，访问a[n]会引起程序崩溃
    //char a[SIZE]//非gcc编译器使用这个
    printf("请输入字符: ");
    getchar();//消除回车'\n'，字符串读入不会读入‘\n‘这行可忽略。
    scanf("%s", a); //字符串读入 自动在后面加入'\0'
    //数组名在这不用加& 原因指针那一章会说明
```

```
    /*for (int i = 0; i < n; i++)
    {
    scanf("%c", &a[i]);
    }//单个字符循环读入*/
    a[n] = '\0'; //字符串结尾
    printf("你输入的字符串为：%s", a);
}
```

> 运行结果：
>
> 请输入字符个数:10
>
> 请输入字符：1234567890
>
> 你输入的字符串为：1234567890

多维数组

定义二维数组

数据类型 数组名[行个数][列个数]

例如：

```
char b[ 2 ][ 3 ] ;     //创建二行三列的二维字符型数组
```

此时b数组中的元素：

b[0][0]	b[0][1]	b[0][2]
b[1][0]	**b[1][1]**	**b[1][2]**

应用二维数组的元素

参考一维数组

二维数组的初始化

```
char b[2][3]={{1,2,3},{4,5,6}};
```

或者

```
char b[2][3]={1,2,3,4,5,6};
```

与下面这句定义等价

```
char b[][3]={1,2,3,4,5,6};//但代表列的方括号内不可省略
```

初始化后有b[0][0]=1, b[0][1]=2, b[0][2]=3, b[1][0]=4, b[1][1]=5, b[1][2]=6

举例说明&更高维数组

一个班里有50位同学，每名同学有语数英三科的成绩，那么我们要如何储存呢？这时候就用到二维数组了。顾名思义，二维数组就是有两个下标的数组，比如：

```
int a[50][2];
```

那么“a[0][0]”就表示第一位同学的语文成绩，而“a[9][2]”就表示第10位同学的英语成绩。 由此推断，同样存在三
维、四维乃至多维数组。 特别地，二维数组由于其表示方法的原因，在很多时候也用来表示矩阵，或与矩阵类
似的结构。在进行类似的表示的时候，我们可以认为第一个维度表示行，第二个维度表示列，比如：

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

是不是与矩阵很像？

程序举例

将一个二维数组行和列的元素互换，存到另一个二维数组中。

```
#include <stdio.h>
int main()

{
    int a[2][3] = {1, 2, 3, 4, 5, 63};
    int b[3][2], i, j;
```

```
    printf("数组a为:\n");
    for (i = 0; i < 2; i++) // 处理a数组中各行元素
    {
        for (j = 0; j < 3; j++) // 处理a数组中各列元素
        {
            printf("%d ", a[i][j]); // 输出a数组的一个元素
            b[j][i] = a[i][j];      // 同时将a数组元素的值赋给b数组的相应元素
        }
        printf("\n");// 添加一个回车
    }
    printf("数组b为:\n");
    for (i = 0; i < 3; i++) // 处理b数组中各行元素
    {
        for (j = 0; j < 2; j++)  // 处理b数组中各列元素
            printf("%d ", b[i][j]); // 处理b数组的一个元素
        printf("\n");
    }
}
```

字符与字符串

那么经过上述的学习呢，大家是不是已经对数字的处理方式比较熟悉了？
但事实上C语言可以也可以对字符进行处理。

> 📌 什么是字符?
> 一个字符可以是一个中文汉字、一个英文字母、一个阿拉伯数字、一个标点符号、一个图形符号或者控
> 制符号等。 在C语言中，我们一般这么定义字符变量:

```
1 char a='?';
```

定义字符型变量的关键字叫做"char"，而且字符变量都是用一对单引号包裹起来的。
这是为了将字符型的数字与表示整型或浮点型的数字做区别，比如：

```
1 char a='1';
2 int b=1;
```

但与此同时，字符型的变量之间也可以互相进行数学运算。这就不得不提到一个叫做ASCALL码的东西。

ASCII编码

上个世纪60年代，美国制定了一套字符编码规则，对英语字符与二进制位之间的关系做了统一规定，这编码规
则被称为ASCII编码，一直沿用至今。
第0～32号及第127号（共34个）是控制字符或通讯专用字符，如控制符：LF（换行）、CR（回车）、FF（换
页）、DEL（删除）、BEL（振铃）等；通讯专用字符：SOH（文头）、EOT（文尾）、ACK（确认）等；
总之就是上面这些没啥卵用。
第33～126号（共94个）是字符，其中第48～57号为0～9十个阿拉伯数字；65～90号为26个大写英文字母，97
～122号为26个小写英文字母，其余为一些标点符号、运算符号等。 具体的编码表格大家需自行查询。

好，那么有了这个表之后，我们愉快的发现，我们有了一条途径把字符和数字互相转换，比如：

```
1 char X='a';
2 char Y=97;
```

以上这两句程序都是把等号左边的变量赋为字符‘a’，而97就是ascall码表中小写字母a的序号。
同理，这个序号我们可以使用整形变量来替代，比如：

```
1 char a='a';
2 int b=97;
```

```
3 char c=97;
```

以上也是合法的。 而在输入输出函数中，字符变量的控制符是“%c”。 同时，字符之间是可以相加减的，比如：

```
1 int a = 'b'-'a';
```

如果此时输出“a”的值会发现程序输出了“1”。 这代表着字符“b”与字符“a”在ascall码表中相差了一位。
幸运的是，大家可以稍微仔细看一下ascall码表，其中阿拉伯数字、大写字母、小写字母之间都是连续排列的，
这是符合大家的普遍认知的。
此外这里需要大家自学一下关于转义字符的相关内容。

字符数组

与其他变量一样，字符型变量也是有数组的，其定义方式与其他类型一样，比如：

```
1 char a[20]={0};
```

但字符数组还可以用来表示字符串！

> 📌 什么是字符串?
> 字符串是由字符组成的一串字符

通俗来说，如果把字符看做字母的话，字符串就是一句话，它用双引号来表示，我们可以通过字符型数组表示
字符串，比如：

```
1 char a[20]="hello,world!";
```

具体来说，字符串与字符的区别和联系可以很好地通过以下两个表达式了解：

```
1 char site[] = {'G','K','D', '\0'};
2 char site[] = "GKD";
```

以上二者是等价的，其中"\0"叫做字符串的结束符，字符串的结尾一定会有这个字符，但是我们在定义字符串
的时候不需要显示的定义它，编译器会帮我们做的。

其实C语言内置了很多关于字符串处理的函数，非常方便，但是在我们领域用到的不太多，所以就不展开讲
了。

作业

- 创建一个有十个元素的一维数组。只赋一个元素的值，使第四个元素的值为4。打印出整个数组。（8分）
- 创建一个三行四列的二维数组，只赋一个元素的值，使第二行第三列的值为8。打印出整个数组。（8分）
- 创建一个有十个元素的一维数组x和一个有五行二列的二维数组y，并按从左到右、从第一行到第五行的顺序将一维数组x的值分别赋值到二维数组y。打印出二维数组y。（8分）
- 仿照一维数组的程序举例，写出一个二维数组的程序，要求如下：
  - 用户可以输入行数列数创建对应数组（3分）
  - 用户可以对二维数组中指定位置赋值（3分）
  - 赋值两回后，打印出当前状态的二维数组（4分）
- 在一个名为time的数组中同时存储小明，小红，诸葛大壮三位同学的早饭、午饭、晚饭的用餐时间。要求使用二维数组，将每一行存储不同的学生信息，其中首列为姓名，后面三列分别为早饭、午饭、晚饭的用餐时间。打印该数组。（10分）

拓展题

- 有一篇文章，共有6行文字，每行有66个字符。分别统计出其中英文大写字母、小写字母、数字、空格以及其它字符的个数。（10分）

函数

作用

使程序更加的整齐规范，同时也可以减少许多重复的代码，模块化程序设计的思路

之前我们接触过最基本的函数就是 “scanf” 函数与 “printf” 函数，有时候我们需要重复执行一些想要的功能，但是C语言自带的函数中没有我们需要的函数（C语言的库函数），我们就需要定制一些自己需要的功能。

比如，判断一个年份是不是闰年，显然这个功能是不会包含在C语言的函数库中的，所以我们需要自己写一个。

（想要程序做什么就设计一个函数去完成。有点类似diy的意思，函数名就类似于这个事情的统一名称）

如何定义

- 函数名称（名字）
- 函数的类型
- 参数（可以为空）
- 函数的主体（要做的事情）

```
类型名 函数名（传入的参数）
{
函数体
}
```

或

```
类型名 函数名（void）    //没有参数传入
{
函数体
}
```

例如：

形参跟实参

形参

全称为"形式参数" 由于它不是实际存在变量，所以又称虚拟变量。是在定义函数名和函数体的时候使用的参数,目的是用来接收调用该函数时传入的参数.在调用函数时，实参将赋值给形参。因而，必须注意实参的个数，类型应与形参一一对应，并且实参必须要有确定的值。

```
int func(a,b);
//这里的a和b就是形式参数，形参。
```

实参

全称为"实际参数"是在调用时传递给函数的参数. 实参可以是常量、变量、表达式、函数等， 无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值， 以便把这些值传送给形参。 因此应预先用赋值，输入等办法使实参获得确定值。

```
int main()
{
    func(1,2);
    return 0;
}
//这里的1和2就是实际参数
```

函数调用时的数据传递

值传递

```c
//值传递

#include <stdio.h>
/* 变量x、y为Swap函数的形式参数 */
void Swap(int x, int y)
{
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    printf("x = %d, y = %d\n", x, y);
}
int main(void)
{
    int a=10;
    int b=20;
     /*变量a、b为Swap函数的实际参数*/
    Swap(a, b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}

//运行结果
x = 20, y = 10
a = 10, b = 20
```

执行Swap函数得到的结果

main函数中的a,b的值不会因运行了Swap函数而改变数值

地址传递

```c
//地址传递
void Swap(int *px, int *py)
{
    int tmp;
    tmp = *px;
    *px = *py;
    *py = tmp;
    printf("*px = %d, *py = %d\n", *px, *py);
}
int main(void)
{
    int a=10;
    int b=20;
    Swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}

//运行结果
x = 20, y = 10
a = 20, b = 10
```

将地址传入函数

Swap函数的执行结果

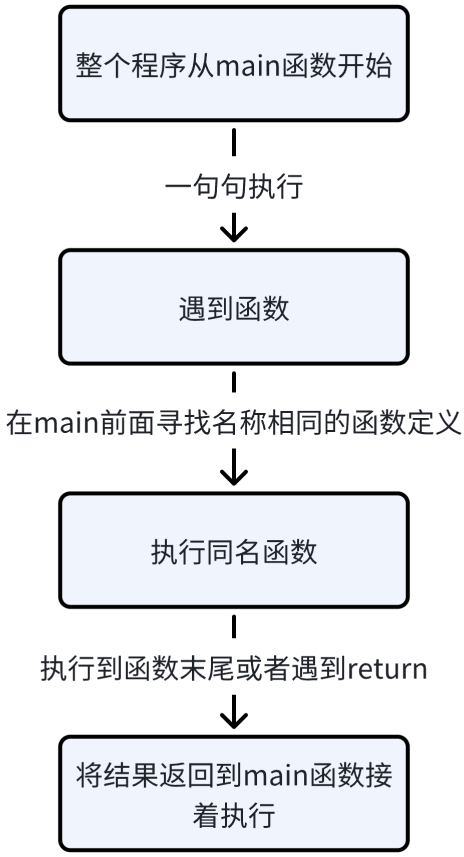
main函数中的a,b变量因执行了Swap函数后，值发生变化

函数作为参数

详见回调函数，在指针章节会详细讨论。

函数的实现原理和函数的声明与调用

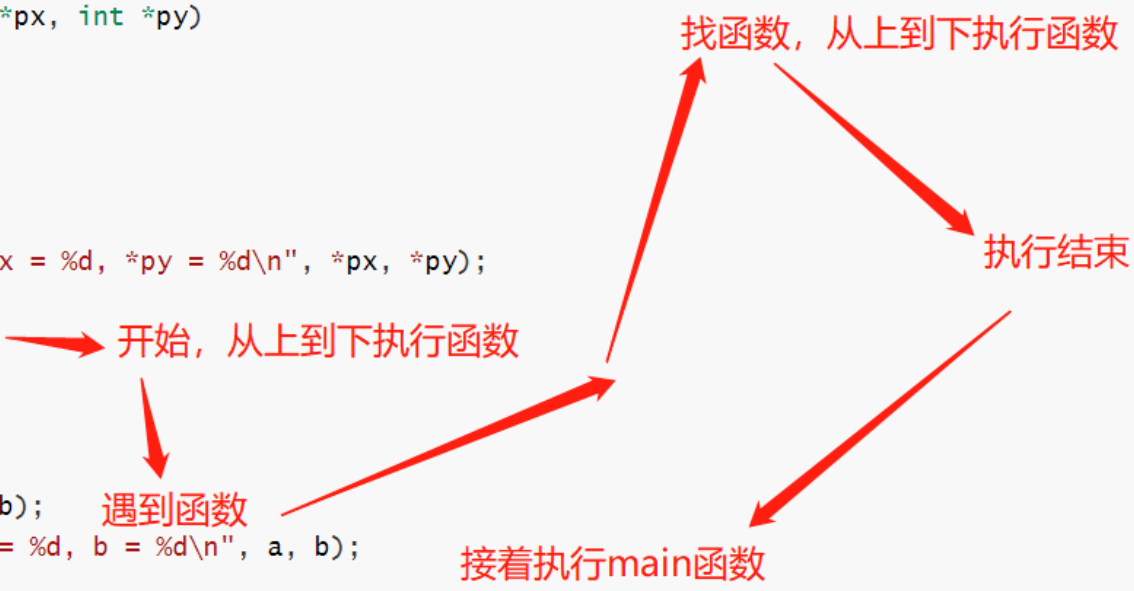
实现原理



即：

```c
//地址传递
void Swap(int *px, int *py)
{
    int tmp;
    tmp = *px;
    *px = *py;
    *py = tmp;
    printf("*px = %d, *py = %d\n", *px, *py);
}
int main(void)
{
    int a=10;
    int b=20;
    Swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}

//运行结果
x = 20, y = 10
a = 20, b = 10
```



函数的声明

方式：函数类型 函数名（参数类型1 参数名1，参数类型2 参数名2，…，参数类型n 参数名n）；

作用：告诉计算机会有这个函数，先不要报错，这个函数在main函数后面会定义

函数的调用

方式：函数名（ 参数名1， 参数名2，…， 参数名n）；

作用：告诉计算机我要用这个函数

```c
//地址传递

int main(void)
{
    void Swap(int *px, int *py);
    int a=10;
    int b=20;
    Swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
void Swap(int *px, int *py)
{
    int tmp;
    tmp = *px;
    *px = *py;
    *py = tmp;
    printf("*px = %d, *py = %d\n", *px, *py);
}

//运行结果
x = 20, y = 10
a = 20, b = 10
```

函数的声明

函数的调用

函数的嵌套

俗称：套娃

函数中有另一个函数即为函数的嵌套

```c
//地址传递
void Swap(int *px, int *py)
{
    int tmp;
    tmp = *px;
    *px = *py;
    *py = tmp;
    printf("*px = %d, *py = %d\n", *px, *py);
}
int main(void)
{
    void Swap(int *px, int *py);
    int a=10;
    int b=20;
    Swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}

//运行结果
x = 20, y = 10
a = 20, b = 10
```

main函数中有Swap函数

注：函数得先定义再嵌套，不然函数就要先声明函数（就是你得嵌套已经知道名字和内容的函数，如果还没有名字和内容的就要先告诉计算机，会有的，只是在后面）

C语言中**不允许作嵌套的函数实现。** 因此各函数之间是平行的，不存在上一级函数和下一级函数的问题。 但是C语言允许在一个函数的定义中出现对另一个函数的调用。

数组作为函数参数

传递数组名实际就是地址传递，main函数中的值会发生改变，具体在指针章节会详细解释。

作业

1. 写一个函数找出两个整数最大值（5分）
2. 写一个函数交换两个整形变量的内容（5分）
3. 写一个函数，使给定的一个3*3的二位整型数组转置，即行列互换（10分）
4. 写一个函数，将两个字符串连接（5分）
5. 写一个函数，输入一个十六进制数，输出相应的十进制数（8分）

结构体

引言

相信师弟师妹们在前面已经学会了整型（int），浮点型（flaot，double），字符型（char），还了解了数组（存储一组具有相同类型的数据），字符串。

但是在实际问题中只有这些数据类型是不够的，有时候我们需要其中的几种一起来修饰某个变量。

想一下，在仲恺，哪些信息能够让别人知道你是谁？没错，就是你的学号（字符串），学院（字符串），专业班级（字符串），姓名（字符串），年龄（int）。

例如，202221724101，自动化学院，自动化221，王远深。

这些数据类型都不同但是他们又是表示一个整体，要存在联系，那么我们就需要一个新的数据类型：结构体。

结构体基本概念

结构体属于用户自定义的数据类型，允许用户通过一个总的变量名存储不同的数据类型（例如前面提到的一个学生（一个变量名）的很多信息（不同的数据类型））

结构体定义

语法：struct 结构体名 { 结构体成员列表 };

例如：

```
struct student
{
    char ID[20]=”202221724101”;
    char College[20]={”自动化学院”};
    char class[20]={”自动化221”};
    char name[20]={”王远深”};
    int age=20;
}
```

接下来讲解一下各个语句的意思

定义结构体时的关键字是struct，不可省略

这个结构体的名字为student

在结构体里面定义成员学号，且类型为字符串型

在结构体里面定义成员年龄，且类型为int

```
//结构体变量创建方式1
struct student stu1; //struct 关键字可以省略
```

可以看出这里的student就是上面我们定义的结构体名

用结构体名定义一个变量为stu1；

结构体成员赋值：

```
stu1.ID = {"202221724101"};
stu1.College = {”自动化学院”};
stu1.name= {”王远深”};
stu1.age=20;
```

那用printf怎么输出呢?

结构体成员的引用

```
printf(“学号为：%s，
学院为：%s，
姓名为：%s，
年龄为：%d.”,stu1.ID,stu1.College,stu1.name,stu1.age);
```

注意：

- **结构体成员的引用格式是结构体名.成员名**
- **结构体成员的引用时候注意对应变量类型，**

结构体嵌套结构体

作用： 结构体中的成员可以是另一个结构体

例如：每个老师辅导一个学员，一个老师的结构体中，记录一个学生的结构体

```
// 学生结构体定义
struct student
{
  // 成员列表
  char name[20]; // 姓名
  int age;     // 年龄
  int score;   // 分数
};
// 教师结构体定义
struct teacher
{
  // 成员列表
  int id;              // 职工编号
  string name;         // 教师姓名
```

```
    int age;            // 教师年龄
    struct student stu; // 子结构体 学生
};
```

那接下来就可以这样赋值

```
struct teacher t1;
t1.id = 10000;
t1.name = "老王";
t1.age = 40;

t1.stu.name = "张三";
t1.stu.age = 18;
t1.stu.score = 100;
```

作业

- 现在学校要综测，每个学生的信息包括学号，姓名，分数（保留两位小数），年龄。（8分）
- 请你定义一个结构体，并赋值，最后输出。（5分）

枚举类型

枚举是C语言中一种基本数据类型，它可以让数据简洁和易读

为什么要用枚举?

```
int MON  1
int TUE  2
int WED  3
int THU  4
int FRI  5
int SAT  6
int SUN  7
```

例如,我们要定义一个星期中的星期几英文对应的阿拉伯数字

以上这些代码，看起来很多很复杂

如果用枚举的方式：

定义格式为：

```
enum 枚举名{
    元素1,
    元素2
};
```

例如：

```
enum DAY{
    MON=1,
    TUE,
    WED,
    THU,
    FRI,
    SAT,
    SUN
};
```

注意：第一个枚举成员的默认值为整型的 0，后续枚举成员的值在前一个成员上加 1。我们在这个实例中把第一个枚举成员的值定义为 1，第二个就为 2，以此类推

枚举变量的定义

```
#include <stdio.h>
enum DAY
{
      MON=1, TUE, WED, THU, FRI, SAT, SUN
};
int main()
{
    enum DAY day;
    day = WED;
    printf("%d",day);
    return 0;
}
```

作业

- 定义一个枚举类型，数据为红橙黄绿青蓝紫，依次代表1~8？试着依次输出他们。（10分）

拓展了解

共用体

含义

在进行某些算法的C语言编程的时候，需要使几种不同类型的变量存放到同一段内存单元中。也就是使用覆盖技术，几个变量互相覆盖。这种几个不同的变量共同占用一段内存的结构，在C语言中，被称作“共用体”类型结构，简称共用体，也叫联合体。

跟结构体有点类似，但是储存方式不一样，结构体的各个成员会占用不同的内存，互相之间没有影响；而共用体的所有成员占用同一段内存，修改一个成员会影响其余所有成员。

如何定义

定义共用体变量一般形式为：

```
union 共用体名
{
    成员表列
}变量表列;
```

储存方式

定义一个共用体

```
union Data
{
int i; //表示不同类型的变量i，ch，f可以存放到同一段存储单元中
char ch;
float f;
}a, b, c; //变量
```

地址空间的表示如下图：

地址	1000	1001	1002	1003
int				
char				
float				

以上3个变量在内存中占的字节数不同，但都是从同一地址开始（图中设为1000）存放，也就是使用覆盖技术，后一个数据覆盖了前面的数据。

共用体变量的大小一般是共用体中成员所占内存最大的成员内存。例如上面的共用体内存就为4个字节。

特点

1. 同一内存段可以用来存放几种不同类型的成员，**但在每一瞬间只能存放其中一个成员，而不是同时存放几个。**
2. 改变其中一个成员，会影响到其他的成员。
3. 共用体变量的地址和它的个成员的地址都是同一个地址。例如：&a.i，&a.ch，&a.f都是同一值。

更多可以参考这个链接中的资料

（https://blog.csdn.net/qq_46359697/article/details/108685538）

位域

概念

有些数据在存储时并不需要占用一个完整的字节，只需要占用一个或几个二进制位即可。例如开关只有通电和断电两种状态，用0和1表示足以，也就是用一个二进位。正是基于这种考虑，C语言又提供了一种数据结构，**叫做“位域”或“位段”。**

定义

位域通过一个结构声明来建立：该结构声明为每个字段提供标签，并确定该字段的宽度。**位域只能是int、unsigned int、signed int类型。**例如，下面的声明建立了个4个1位的字段：

```
struct
{
   unsigned int autfd:1;
   unsigned int bldfc:1;
   unsigned int undin:1;
   unsigned int itals:1;
}prnt;
```

根据该声明， prnt包含4个1位的字段。现在，可以通过普通的结构成员运算符(.)单独给这些字段赋值：

```
prnt.itals = 0:
prnt.undin = 1;
```

由于每个字段恰好为1位，所以只能为其赋值1或0。变量prnt被储存在int大小的内存单元中，但是在本例中只使用了其中的4位。

: 后面的数字用来限定成员变量占用的位数。位域的宽度不能超过它所依附的数据类型的长度。通俗地讲，成员变量都是有类型的，这个类型限制了成员变量的最大长度，: 后面的数字不能超过这个长度。

如上述结构中autfd、bldfc、undin、itals后面的数字不能超过unsigned int的位数，即在32bit环境中就是不能超过32。

超过这个这个限制，就会发生上溢（溢出的一种），可以去课外了解了解。

使用情况

1. 当机器可用内存空间较少而使用位域可以大量节省内存时。如,当把结构作为大数组的元素时。
2. 当需要把一结构或联合映射成某预定的组织结构时。例如,当需要访问字节内的特定位时。