# 目录

[目录 1](#_Toc8177325)

[第9章 C语言-联合体&枚举 2](#_Toc8177326)

[9.1 C语言联合体概述 2](#_Toc8177327)

[9.2 联合体变量的定义与初始化 2](#_Toc8177328)

[9.2.1 联合体变量定义 2](#_Toc8177329)

[9.3 联合体变量的使用 2](#_Toc8177330)

[9.4 联合体数组的定义与初始化 2](#_Toc8177331)

[9.5 联合体数组的使用 2](#_Toc8177332)

[9.6 枚举概述 2](#_Toc8177333)

[9.7 枚举定义 2](#_Toc8177334)

[9.8 枚举使用 2](#_Toc8177335)

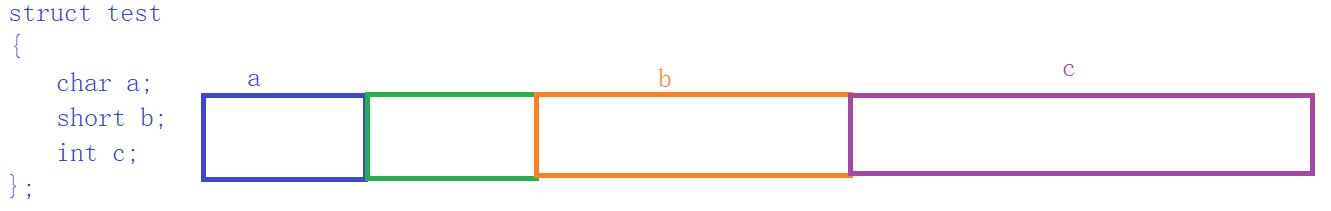
# C语言-联合体&枚举

## C语言联合体概述

结构体：具有相同数据类型或者不同数据类型的数据组成的集合

每个成员都有自已一个独立的空间

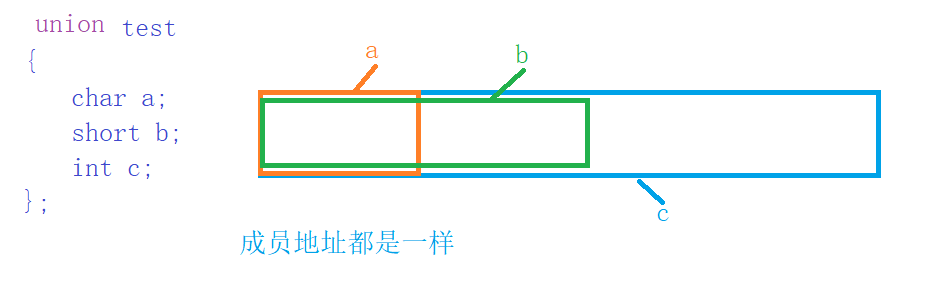
结构体所占空间大小>所有成员所占空间之和



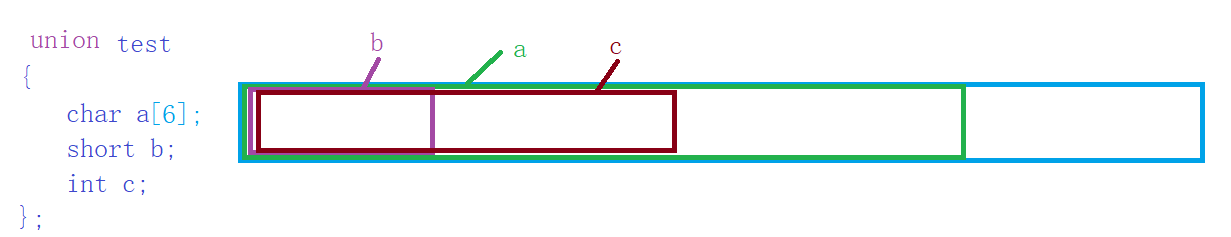
共用体：具有相同数据类型或者不同数据类型的集合

每个成员共用内存空间，空间大小由成员中数据类型最大的来进行分配

union：共用体关键字



如果共用体成员有数组类型，那么空间大小由数据类型最大的成员的整数倍



节约空间

学校管理系统(学生和老师)

struct

{

姓名;

性别；

年龄：

union //节省内存空间 共用体

{

绩效；

成绩；

}

}

学生：成绩

老师：绩效

## 共用体变量的定义与初始化

### 构造共用体类型

共用体在使用前先构造出共用体类型

|  |
| --- |
| union 共用体名  {  成员列表；  数据类型 成员名1；  数据类型 成员名2；  数据类型 成员名3；  数据类型 成员名4；  ...  };分号不能省略  struct 结构体名：结构体类型  union 共用体名：共用体类型 |

### 联合体变量定义

上面已将共用体类型定义好了，接下来就可以用共用体类型来定义变量

定义共用体变量也有三种形式

1. 构造共用体类型的同时定义共用体变量
2. 先构造共用体类型，再定义共用体变量
3. 构造共用体类型的同时定义共用体变量，但是省略共用体名
4. 升级改造：typedef给数据类型取别名

### 共用体变量的初始化

union 共用体名 变量名 = {0};

## 联合体变量的使用

格式：共用体名.成员名

1. 变量的成员地址都是一样
2. 共用体变量的值，是最后一次赋值给共用体成员的值，也就是后面赋值会覆盖前面赋的值

|  |
| --- |
| #include <stdio.h>  typedef union Test  {  char a;  short b;  int c;  }TEST;  int main(void)  {  TEST t1 = {'a'};  t1.a = 'a';  t1.b = 10;  t1.c = 55;    printf("%c\r\n", t1.a);  printf("%d\r\n", t1.b);  printf("%d\r\n", t1.c);  printf("%#p\r\n", &t1.a);  printf("%#p\r\n", &t1.b);  printf("%#p\r\n", &t1.c);  return 0;  } |

## 联合体数组的定义与初始化

格式：union 共用体名 数组名[长度]；//定义了一个共用体数组

初始化：union 共用体名 数组名[长度] = {0}；

## 联合体数组的使用

格式：数组名[下标].成员名

|  |
| --- |
| typedef union Test  {  char a;  short b;  int c;  }TEST;  int main(void)  {  TEST t1[3] = {'a', 10, 55};    printf("%c\r\n", t1[0].a);  printf("%c\r\n", t1[1].a);  printf("%c\r\n", t1[2].a);  return 0;  } |

## 枚举概述

在C语言中：将一件事物所有的发生情况一一列举出来

一个星期：7天

一年：四季

枚举也是一种构造数据类型，使用前也要构造出来

枚举成员：枚举子(整型)

作用：方便程序员阅读程序

关键字：enum

注意：成员之间用逗号隔开，每个成员都是整型

如果没有对成员进行赋值，那么第一个成员的值就是0

## 枚举定义

格式：enum 枚举名 变量名；

## 枚举使用

只能是枚举子

把成员名当成宏名来使用

|  |
| --- |
| #include <stdio.h>  enum Week  {  SUN,  MON,  TUE,  WED,  THU,  FRI,  SAT  };  int main(void)  {  enum Week w;//定义了一个枚举类型变量， 一般是枚举子  w = 10;  printf("w = %d\r\n", w);  return 0;  } |

## 内存分配

1. 自动分配(编译器)：定义变量的这个形式就叫自动分配 int a;

数据类型 变量名；数据类型：决定空间的大小和空间的存储格式

优点：方便、直接

缺点：一旦分配，大小是固定，中途是不能更改

数组：数据类型 数组名[长度]；

系统根据元素类型和元素个数来进行分配空间，空间就固定

int a[5];

元素类型是int类型，元素个数是5个的数组a

一共20字节空间大小

删除和插入：通过覆盖或者移开等这些操作

存1个数据，空间浪费

存6个数据，空间不足

1. 动态分配(程序员)：由程序员来申请的空间，空间大小和空间类型由程序决定

程序员申请空间 🡪 通过内存管理函数

8字节 🡪 空间作为什么来使用，由程序员决定

## 内存管理函数

1. malloc函数
2. 头文件：#include <stdlib.h>
3. 函数功能：申请size个字节空间，返回空间首地址
4. 函数原型：void \*malloc( size\_t size );//16
5. 函数形参：

void \*：可以指向任意类型的指针 🡪 万能指针

size\_t：unsigned int

size：申请的字节数

1. 返回值：申请成功会返回该空间的首地址，申请失败，返回NULL，可以用它判断
2. 注意：malloc申请的空间里面的值是随机值；
3. calloc函数
4. 头文件：#include <stdlib.h>
5. 函数功能：申请num块空间，每一块空间size字节
6. 函数原型：void \*calloc( size\_t num, size\_t size );//4 4 1 16
7. 函数形参：

size\_t num：申请多少块

size\_t size：一块有多多少字节

1. 返回值：申请成功会返回该空间的首地址，申请失败，返回NULL，可以用它判断
2. 注意：calloc申请的空间，系统会将它的空间初始化为0；
3. free函数
4. 头文件：#include <stdlib.h>
5. 函数功能：释放程序员申请的空间
6. 函数原型：void free( void \*ptr );
7. 函数形参：

void \*ptr：指向需要释放空间的首地址

1. 注意：

程序员申请的空间，不会自动释放，人为释放它

如果不释放，程序结束之后系统会进行回收

内存管理函数：内存遗漏和内存重写

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main(void)  {  char ch;  char \*f = (char \*)calloc(2, 4);    // f = &ch;//内存遗漏  free(f);//内存重写  \*f = 'b';  printf("%d\r\n", \*f);  printf("%d\r\n", \*(f+1));  printf("%d\r\n", \*(f+2));  printf("%d\r\n", \*(f+3));  printf("%d\r\n", \*(f+4));  return 0;  } |