一、结构体类型的概念及定义

1.1 基本概述

构造类型：

不是基本类型的数据结构也不是指针，它是若干个相同或不同类型的数据构成的集合

常用的构造类型有数组、结构体、共用体

数组用于保存多个相同类型的数据

结构体用于保存多个不同类型的数据

1.2 结构体的概念

结构体是一种构造类型的数据结构，

是一种或多种基本类型或构造类型的数据的集合。

1.3 结构体类型的定义

1.3.1 先定义结构体类型，再去定义结构体变量

struct 结构体类型名{

成员列表

};

例子：

```
struct stu{
 int num;
 char name[20];
 char sex;
};
```

//有了结构体类型后，就可以用类型定义变量了

struct stu lucy, bob, lilei; //定义了三个struct stu类型的变量，注意不要把struct省略

每个变量都有三个成员，分别是num name sex

1.3.2 在定义结构体类型的时候顺便定义结构体变量，以后还可以定义结构体变量

struct 结构体类型名{

成员列表;

}结构体变量1,变量2;

struct 结构体类型名 变量3，变量4；

例子：

```
struct stu{
```

```
 int num;
 char name[20];
 char sex;
}lucy,bob,lilei;
struct stu xiaohong,xiaoming;
```

注意：一般结构体类型都会定义在全局，也就是main函数的外面

所以在定义结构体类型的同时定义变量，这些变量一般都是全局变量

定义完类型之后定义的结构体变量内存分配要看定义的位置

1.3.3 无名结构体的定义

在定义结构体类型的时候，没有结构体类型名，顺便定义结构体变量，

因为没有类型名，所以以后不能再定义相关类型的数据了

struct {

成员列表;

}变量1，变量2;

注意：无名结构体由于没有结构体名，所以定义完之后是无法在定义结构体变量的，只能在定义类型的同时定义结构体变量

例子：

```
struct {
 int num;
 char name[20];
 char sex;
}lucy,bob;
```

1.3.4 给结构体类型取别名

通常咱们将一个结构体类型重新起个类型名，用新的类型名替代原先的类型

typedef struct 结构体名 {

成员列表;

}重新定义的结构体类型名A;

注意：typedef主要用于给一个类型取别名，此时相当于给当前结构体重新起了一个类型名为A，

相当于 struct 结构体名，所以如果结构体要取别名，一般不需要先给结构体定义名字，定义结构体变量时，直接使用A即可，不用加struct

例子：

```
typedef struct stu{
 int num;
 char name[20];
 char sex;
}STU;
```

以后STU 就相当于 struct stu

STU lucy;和struct stu lucy;是等价的，所以可以不指定stu这个名字

二、结构体变量的定义初始化及使用

2.1 结构体变量的定义和初始化

结构体变量，是个变量，这个变量是若干个数据的集合

注：

(1):在定义结构体变量之前首先得有结构体类型，然后在定义变量

(2):在定义结构体变量的时候，可以顺便给结构体变量赋初值，被称为结构体的初始化

(3):结构体变量初始化的时候，各个成员顺序初始化

案例：

```
#include <stdio.h>

//定义结构体类型
struct stu{
 int id;
 char name[32];
 char sex;
 int age;
//定义结构体变量之定义结构体类型的同时定义结构体变量
}zhangsan, lisi = {1002, "李四", 'B', 25};

//使用typedef对结构体类型取别名
typedef struct{
 int a;
 int b;
 char c;
}MSG;

int main(int argc, char *argv[])
{
```

```
 //定义结构体变量之类型定义完毕之后定义变量
 struct stu wangwu;
 //结构体变量的初始化
 struct stu zhaoliu = {1001, "赵六", 'B', 20};

 //如果使用typedef对结构体类型取别名
 //就无法在定义类型的同时定义结构体变量
 //在定义结构体变量的时候不用加struct
 MSG msg1, msg2 = {100, 200, 'w'};

 return 0;
}
```

2.2 结构体变量的使用

结构体变量对成员调用的方式：

```
结构体变量.结构体成员
```

注意：这地方说的结构体变量主要指的是普通结构体变量

2.2.1 结构体变量的简单使用

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

struct stu{
 int id;
 char name[32];
 char sex;
 int age;
}zhangsan, lisi = {1002, "李四", 'B', 25};

typedef struct{
 int a;
 int b;
 char c;
}MSG;

int main(int argc, char *argv[])
{
 struct stu wangwu;
 struct stu zhaoliu = {1001, "赵六", 'B', 20};

```

```
 MSG msg1, msg2 = {100, 200, 'w'};

 //结构体变量的使用
 zhangsan.id = 1001;
 strcpy(zhangsan.name, "张三");
 zhangsan.sex = 'B';
 zhangsan.age = 18;
 printf("%d - %s - %c - %d\n", zhangsan.id, zhangsan.name,\
 zhangsan.sex, zhangsan.age);

 printf("%d - %s - %c - %d\n", lisi.id, lisi.name,\
 lisi.sex, lisi.age);

 printf("%d - %s - %c - %d\n", zhaoliu.id, zhaoliu.name,\
 zhaoliu.sex, zhaoliu.age);

 printf("%d - %d - %c\n", msg2.a, msg2.b, msg2.c);

 return 0;
}
```

执行结果

```
Debug\debug\02_struct_use.exe.
1001 - 张三 - B - 18
1002 - 李四 - B - 25
1001 - 赵六 - B - 20
100 - 200 - w
C:\Users\lzx\Desktop\src\build
```

2.2.2 在结构体中嵌套结构体

```
//在结构体中嵌套结构体
typedef struct{
 int year;
 int month;
 int day;
}BD;

typedef struct{
 int id;
 char name[32];
 BD birthday;
}STU;
```

```
13
14 void test2()
15 {
16  STU xiaoming;
17  xiaoming.id = 1001;
18  strcpy(xiaoming.name, "小明");
19  //如果结构体中嵌套结构体，赋值时找到最内层的成员再进行赋值
20  xiaoming.birthday.year = 2002;
21  xiaoming.birthday.month = 12;
22  xiaoming.birthday.day = 20;
23
24  printf("%d - %s - %d-%d-%d\n", xiaoming.id, xiaoming.name,\
25  xiaoming.birthday.year, xiaoming.birthday.month,\
26  xiaoming.birthday.day);
27
28  //嵌套的形式定义并初始化
29  STU xiaoli = {1002, "小丽", 2000, 1, 1};
30  printf("%d - %s - %d-%d-%d\n", xiaoli.id, xiaoli.name,\
31  xiaoli.birthday.year, xiaoli.birthday.month,\
32  xiaoli.birthday.day);
33 }
```

执行结果

```
Debug\debug\02_struct_use.exe...
1001 - 小明 - 2002-12-20
1002 - 小丽 - 2000-1-1
C:\Users\lzx\Desktop\src\build-
```

2.3 相同类型的结构体变量可以相互赋值

```
1 #include <stdio.h>
2 #include <string.h>
3
4 struct stu{
5  int id;
6  char name[32];
7  char sex;
8  int age;
9 };
10
11 int main(int argc, char *argv[])
12 {
```

```
 struct stu zhangsan;
 zhangsan.id = 1001;
 strcpy(zhangsan.name, "张三");
 zhangsan.sex = 'B';
 zhangsan.age = 18;
 printf("%d - %s - %c - %d\n", zhangsan.id, zhangsan.name,\
 zhangsan.sex, zhangsan.age);


 //相同类型的结构体变量之间可以直接赋值
 struct stu lisi;
 lisi = zhangsan;
 printf("%d - %s - %c - %d\n", lisi.id, lisi.name,\
 lisi.sex, lisi.age);

 return 0;
}

```

执行结果

```
Desktop_Qt_5_8_0_MinGW_32bit-
1001 - 张三 - B - 18
1001 - 张三 - B - 18
C:\Users\lzx\Desktop\src\buil
```

三、结构体数组

结构体数组是个数组，由若干个相同类型的结构体变量构成的集合

1、结构体数组的定义方法

```
struct 结构体类型名 数组名[元素个数];
```

例子：

```
struct stu{
 int num;
 char name[20];
 char sex;
};
 struct stu edu[3];
 定义了一个struct stu 类型的结构体数组edu，
 这个数组有3个元素分别是edu[0] 、edu[1]、edu[2]
```

2、结构体数组元素的引用

```
数组名[下标]
```

3、结构体数组元素对成员的使用

```
数组名[下标].成员
```

案例：

```c
#include <stdio.h>

typedef struct{
 int num;
 char name[20];
 float score;
}STU;

int main(int argc, char *argv[])
 {
  //定义一个结构体数组
  STU edu[3] = {
  {101,"Lucy",78},
  {102,"Bob",59.5},
  {103,"Tom",85}
  };

  //输出结构体数组中的元素
  int j;
  for(j = 0; j < 3; j++)
  {
  printf("%d - %s - %.2f\n", edu[j].num, edu[j].name,\
  edu[j].score);
  }

  int i;
  float sum=0;
  for(i = 0; i < 3; i++)
  {
  sum += edu[i].score;
  }

  printf("平均成绩为%.2f\n",sum / 3);

```

```
 return 0;
}

```

执行结果

```
Debug\debug\04_struct_array.exe...
101 - Lucy - 78.00
102 - Bob - 59.50
103 - Tom - 85.00
平均成绩为74.17
C:\Users\lzx\Desktop\src\build-04_st
```

四、结构体指针

即结构体的地址，结构体变量存放内存中，也有起始地址

咱们定义一个变量来存放这个地址，那这个变量就是结构体指针变量。

1、结构体指针变量的定义方法：

```
struct 结构体类型名  * 结构体指针变量名;
```

2、结构体指针变量对成员的引用

```
(*结构体指针变量名).成员
结构体指针变量名->成员
```

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

struct stu{
 int id;
 char name[32];
 char sex;
 int age;
};

int main(int argc, char *argv[])
{
 //定义一个结构体指针变量
 struct stu *s;
 //在堆区开辟结构体空间并将其地址保存在结构体指针变量中
 s = (struct stu *)malloc(sizeof(struct stu));

 s->id = 1001;
```

```
20  strcpy(s->name, "张三");
21  s->sex = 'B';
22  s->age = 20;
23
24  printf("%d - %s - %c - %d\n", s->id, s->name, s->sex, s->age);
25
26  return 0;
27 }
```

执行结果

```
Debug\debug\05_struct_poin
1001 - 张三 - B - 20
C:\Users\lzx\Desktop\src\b
```

五、结构体内存分配问题

规则1：以多少个字节为单位开辟内存

给结构体变量分配内存的时候，会去结构体变量中找基本类型的成员

哪个基本类型的成员占字节数多，就以它大大小为单位开辟内存,

在gcc中出现了double类型的例外

(1)：成员中只有char型数据 ，以1字节为单位开辟内存。

(2)：成员中出现了short 类型数据，没有更大字节数的基本类型数据。

以2字节为单位开辟内存

(3)：出现了int 、float 没有更大字节的基本类型数据的时候以4字节为单位开辟内存。

(4)：出现了double类型的数据

情况1：

在vc里，以8字节为单位开辟内存。

情况2：

在gcc里，以4字节为单位开辟内存。

无论是那种环境，double型变量，占8字节。

(5)：如果在结构体中出现了数组，数组可以看成多个变量的集合。

如果出现指针的话，没有占字节数更大的类型的，以4字节为单位开辟内存。

(6)：在内存中存储结构体成员的时候，按定义的结构体成员的顺序存储。

规则2：字节对齐

(1)：char　　1字节对齐 ，即存放char型的变量，内存单元的编号是1的倍数即可。

(2)：short　　2字节对齐 ，即存放short int 型的变量，起始内存单元的编号是2的倍数即可。

(3)：int　　4字节对齐 ，即存放int 型的变量，起始内存单元的编号是4的倍数即可

(4)：long　　在32位平台下，4字节对齐 ，即存放long int 型的变量，起始内存单元的编号是4的倍数即可

(5)：float　　4字节对齐 ，即存放float 型的变量，起始内存单元的编号是4的倍数即可

(6)：double

a.vc环境下

8字节对齐，即存放double型变量的起始地址，必须是8的倍数，double变量占8字节

b.gcc环境下

4字节对齐，即存放double型变量的起始地址，必须是4的倍数，double变量占8字节。

注意：

当结构体成员中出现数组的时候，可以看成多个变量。

开辟内存的时候，从上向下依次按成员在结构体中的位置顺序开辟空间

```
struct stu{
 int num;
 int age;
}lucy;
8字节

struct stu{
 char sex;
 int age;
}lucy;
8字节

struct stu{
 char a;
 short int b;
 int c;
}temp;
8字节

struct stu{
 char a;
 int c;
```

```
 short int b;
}temp;
12字节

struct stu{
 char buf[10];
 int a;
}temp;
16字节

struct stu{
 char a;
 double b;
};
12字节
```

为什么要有字节对齐？

用空间来换时间，提高cpu读取数据的效率

六、位段

一、位段

在结构体中，以位为单位的成员，咱们称之为位段(位域)。

struct packed_data{

unsigned int a:2;

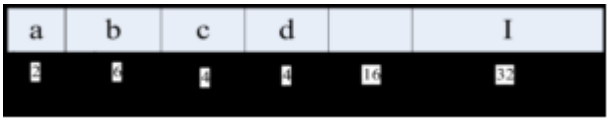
unsigned int b:6;

unsigned int c:4;

unsigned int d:4;

unsigned int i;

} data;



注意：不能对位段成员取地址

```
#include<stdio.h>
struct packed_data{
unsigned int a:2;
unsigned int b:6;
```

```
unsigned int c:4;
unsigned int d:4;
unsigned int i;
} data;
int main()
 {
  printf("%d\n",sizeof(data));
  printf("%p\n",&data);
  printf("%p\n",&(data.i));
  return 0;
 }
```

位段注意：

1、对于位段成员的引用如下：

data.a =2

赋值时，不要超出位段定义的范围;

如段成员a定义为2位，最大值为3,即（11）2

所以data.a =5，就会取5的低两位进行赋值 101

2、位段成员的类型必须指定为整形或字符型

3、一个位段必须存放在一个存储单元中，不能跨两个单元

第一个单元空间不能容纳下一个位段，则该空间不用，

而从下一个单元起存放该位段

位段的存储单元：

(1)：char型位段 存储单元是1个字节

(2)：short int型的位段存储单元是2个字节

(3)：int的位段，存储单元是4字节

(4)：long int的位段，存储单元是4字节

```
struct stu{
char a:7;
char b:7;
char c:2;
}temp;//占3字节，b不能跨 存储单元存储
```

```
#include<stdio.h>
struct stu{
```

```
 char a:7;
 char b:7;
 char c:2;
}temp;
int main()
{
 printf("%d\n",sizeof(temp));
  return 0;
 }
 结果为：3 ， 证明位段不能跨其存储单元存储
```

注意：不能 取 temp.b的地址，因为b可能不够1字节，不能取地址。

4、位段的长度不能大于存储单元的长度

(1)：char型位段不能大于8位

(2)：short int型位段不能大于16位

(3)：int的位段，位段不能大于32位

(4)：long int的位段，位段不能大于32位

```
#include<stdio.h>
struct stu{
 char a:9;
 char b:7;
 char c:2;
}temp;
int main()
{
 printf("%d\n",sizeof(temp));
  return 0;
 }
 分析：
 编译出错，位段a不能大于其存储单元的大小
```

5、如一个段要从另一个存储单元开始，可以定义：

unsigned char a:1;

unsigned char b:2;

unsigned char :0;

unsigned char c:3;(另一个单元)

由于用了长度为0的位段，其作用是使下一个位段从

下一个存储单元开始存放

将a、b存储在一个存储单元中，c另存在下一个单元

```
#include<stdio.h>
struct m_type{
 unsigned char a:1;
 unsigned char b:2;
 unsigned char :0;
 unsigned char c:3;
};
int main()
{
  struct m_type temp;
  printf("%d\n",sizeof(temp));
  return 0;
 }
```

6、可以定义无意义位段,如：

unsigned a: 1;

unsigned : 2;

unsigned b: 3;

```
struct data{
 char a:1;
 char b:1;
 char c:1;
 char d:1;
 char e:1;
 char f:1;
 char g:1;
 char h:1;
}temp;
int main()
{
 char p0;
 //p0=0x01;// 0000 0001
 temp.a=1;
 //p0=temp;//错的，类型不匹配
 //p0=(char)temp;//错的，编译器不允许将结构体变量，强制转成基本类型的。
 p0 = *((char *)(&temp));
```

```
}
```

七、共用体

1：共用体和结构体类似，也是一种构造类型的数据结构。

既然是构造类型的，咱们得先定义出类型，然后用类型定义变量。

定义共用体类型的方法和结构体非常相似，把struct 改成union 就可以了。

在进行某些算法的时候，需要使几种不同类型的变量存到同一段内存单元中，几个变量所使用空

间相互重叠

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

共用体所有成员占有同一段地址空间

共用体的大小是其占内存长度最大的成员的大小

共用体的特点：

1、同一内存段可以用来存放几种不同类型的成员，但每一瞬时只有一种起作用

2、共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员后原有的成员的值会被覆盖

3、共用体变量的地址和它的各成员的地址都是同一地址

4、共用体变量的初始化union data a={123}; 初始化共用体为第一个成员

```c
#include <stdio.h>

//定义一个共用体
union un{
 int a;
 int b;
 int c;
};

int main(int argc, char *argv[])
{
 //定义共用体变量
 union un myun;
 myun.a = 100;
```

```
15   myun.b = 200;
16   myun.c = 300;
17
18   printf("a = %d, b = %d, c = %d\n", myun.a, myun.b, myun.c);
19
20   return 0;
21  }
```

执行结果

```
Starting C:\Users\lzx\Desktop\s
\debug\06_union.exe...
a = 300, b = 300, c = 300
C:\Users\lzx\Desktop\src\build-
\06_union.exe exited with code
```

八、枚举

将变量的值一一列举出来，变量的值只限于列举出来的值的范围内

枚举类型也是个构造类型的

1、枚举类型的定义

```
1  enum 枚举类型名{
2    枚举值列表;
3  };
```

在枚举值表中应列出所有可用值,也称为枚举元素

枚举变量仅能取枚举值所列元素

2、枚举变量的定义方法

```
1  enum 枚举类型名   枚举变量名;
```

① 枚举值是常量,不能在程序中用赋值语句再对它赋值

例如：sun=5; mon=2; sun=mon; 都是错误的.

② 枚举元素本身由系统定义了一个表示序号的数值

默认是从0开始顺序定义为0，1，2...

如在week中，mon值为0，tue值为1， ...,sun值为6

③ 可以改变枚举值的默认值：如

enum week //枚举类型

{

mon=3，tue，wed，thu，fri=4，sat,sun

};

mon=3 tue=4,以此类推

fri=4 以此类推

注意：在定义枚举类型的时候枚举元素可以用等号给它赋值，用来代表元素从几开始编号

在程序中，不能再次对枚举元素赋值，因为枚举元素是常量。

```c
#include <stdio.h>

//定义一个枚举类型
enum week
{
 mon=8, tue, wed, thu=2, fri, sat, sun
};

int main(int argc, char *argv[])
{
  //定义枚举类型的变量
  enum week day = mon;
  printf("day = %d\n", day);

  day = fri;
  printf("day = %d\n", day);

  return 0;
}
```

执行结果

```
Starting C:\Users\lzx\Desktop\src
\debug\07_enum.exe...
day = 8
day = 3
C:\Users\lzx\Desktop\src\build-07
\07_enum.exe exited with code 0
```