



专题九 结构体变量的定义与使用

【内容预览】



【知识清单】

9.1、结构的定义与使用

9.1.1、结构的定义

结构是由不同数据类型的数据组成的。组成结构的每个数据称为该结构的成员项，简称成员。结构的定义是宣布该结构是由几个成员项组成，以及每个成员项具有什么样的数据类型。结构定义的一般形式为：

```
struct 结构名
{
    数据类型  成员名 1;
    数据类型  成员名 2;
    .....
    数据类型  成员名 3;
};
```

例如：

```
struct Employee
{
    char name[20];
    char sex;
    int old;
    int wage;
};
```

该结构的名称是 `Employee`，它由 4 个成员组成。第一个成员项是字符型数据 `name[]`，用于保存姓名字符串；第二个成员项是字符型数据 `sex`，用于保存性别字符；第三个成员项是 `int` 型整数 `old`，用于保存年龄数据；最后一个成员项是 `int` 型整数 `wage`，用于保存工资数据。

1. 结构定义以关键字 `struct` 作为标识符，其后是定义的结构名，二者形成了特定结构的类型标识符。结构名由用户定义，虽然结构体是用户自行定义的新数据类型，但是编译系统把结构名 `Employee` 与 `int`、`double` 等基本数据类型同等对待，即结构名就像基本数据类型一样，用来说明具体的结构变量。

2. 在结构名下面的一对花括号中的是组成该结构的各个成员项。每个成员项由其数据类型和成员名组成。每个成员由其数据类型和成员名组成。**每个成员项后用“；”作为结束符。整个结构的定义也用分号作为结束符。**

3. 结构的定义明确描述了该结构的组织形式。在程序执行时，结构的定义并不引起系统为该结构分配内存，但是还没有占用实际的内存空间。

```
struct Employee
{
    char name[20];    //20 字节
    char sex;         //1 字节
    int old;          //2 字节
    int wage;         //2 字节
};
```

9.1.2、结构变量的定义

定义结构体变量的一般形式为：

<存储类型> struct 结构名 结构变量名

例如：`struct Employee wang;`

由这个说明可以知道，结构变量 `wang` 是由前面所说的 `Employee` 中的四个成员项组成的，每个成员项的类型和名字都与 `Employee` 结构定义中给出的相同。

注意：1. 结构变量一旦定义，系统将会为定义的结构变量分配一定的内存空间。当多个变量使用结构的时候，它们可以一起定义。例如：`struct Employee wang, li, zhang;`

2. 结构变量使用内存空间，所以它们也具有一定的存储类型。

3. 在程序中，定义结构变量要在定义结构之后，**对于尚未定义的结构，不能用它对任何结构变量进行说明。**

4. 结构的定义和结构变量的定义有时候也可以同时进行，在这种情况下，有时会省略结构名。

例如：

```
struct Employee
{
    char name[20];
    char sex;
    int old;
    int wage;
} wang, song, zhou;
```

5. 结构变量占用内存的实际大小，可以用 `sizeof` 运算求出。`sizeof` 运算的功能是计算出给定的运算量占用内存空间的字节数。它的运算表达式一般形式如下：

`sizeof (运算量)`

其中，运算量可以是变量、数组或是结构变量。

9.1.3、结构变量的使用形式和初始化

1. 结构是若干不同数据类型的集合体。在程序中使用结构时，一般情况下不能把结构作为一个整体来参与

数据处理，而参加各种运算和操作的时候结构的各个成员用以下形式表示：

结构变量名. 成员名

当结构成员是指针变量时，要注意它的使用形式上的特点。

例如： struct Employee

```
{
    char *name;
    char sex;
    int old;
    int wage;
} zhou;
```

定义的 Employee 结构中的成员项 name 是一个 char 型指针。如果结构变量 zhou 被说明为 Employee 结构，则 zhou 的成员项 name 是一个 char 型指针。那么 *zhou.name 就表示该指针指向的目标变量，它就等同于 * (zhou.name)。

2. 在结构说明的同时，可以给这个结构的每个成员赋初值，称为结构的初始化。结构初始化的一般形式为：

struct 结构名 结构变量={初始数据}

其中，花括号中包围的初始数据之间用逗号分隔，初始数据的个数与结构成员项的个数应该相同，它们是按照先后顺序一一赋值的。此外，每个数据必须符合与其对应的成员项的数据类型。

例如： struct Employee wang={"wang hai",'M',34,"010-12345678","Beijing"};

它所实现的功能，与下列赋值语句是一样的。

```
wang.name="wang hai";
wang.sex='M';
wang.old=34;
wang.tel="010-12345678";
wang.adr="beijing";
```

下面，举个例子来分析一下：

```
#include<stdio.h>
struct Employee
{
    char *name;
    char sex;
    int old;
    char *tel;
    char *adr;
};
void main()
{
    struct Employee wang,gao; //结构体变量的定义
    wang.name="wang hai";    //结构变量的成员赋值
    wang.sex='M';
    wang.old=34;
    wang.tel="010-12345678";
    wang.adr="beijing";
    gao.name="gao yang";
```

```
gao.sex='F';
gao.old=42;
gao.tel="021-87654321";
gao.adr="shanghai";
}
```

9.2、结构数组与结构指针

9.2.1、结构数组

在 C 语言中，具有相同数据类型的数据可以组成数组，指向相同类型的指针可以组成指针数组。同理，具有相同结构变量的结构也可以组成数组，称为结构数组。

结构数组的说明形式如下：

<存储类型> struct 结构名 结构数组名[元素个数]={初值表};

注意：1.在对结构类型进行初始化时，方括号中的元素个数可以缺省。

2.结构数组也具有数组的属性，结构数组名是结构数组存储首地址。

例：有 3 个候选人，每个选民只能投票选一人，要求编一个统计选票的程序，先后输入被选人名字，最后输出个人得票的结果。

```
#include<stdio.h>
#include<string.h>
struct Person                                //声明结构体类型 struct Person
{
    char name[20];                          //候选人姓名
    int count;                              //候选人得票数
} leader[3]={ "Li",0,"Zhang",0,"Sun",0}; //定义结构体数组并初始化
int main()
{
    char leader_name[20];                   //定义字符数组
    int i,j;
    for(i=1;i<=10;i++)
    {
        scanf("%s",leader_name);          //输入所选的候选人姓名
        for(j=0;j<3;j++)
            if(strcmp(leader_name,leader[j].name)==0)
                leader[j].count++;
    }
    printf("\nResult:\n");
    for(i=0;i<3;i++)
        printf("%5s:%d\n",leader[i].name,leader[i].count);
    return 0;
}
```

在主函数中定义字符数组 `leader_name`，用它存放被选人的姓名。在每次循环中输入一个被选人姓名，将其在结构中进行比较，当找到相同姓名时，`leader[j].count++`，即票数加 1。

9.2.2、结构指针

指向结构的指针变量称为结构指针，它与其他各类指针在特性和使用方法上完全相同。结构指针的运算仍按地址计算规则进行，其定义格式为

〈存储类型〉 struct 结构名 *结构指针名

例如：
 struct Employee *pman;
 struct Employee *pd;

注：1.存储类型是结构指针变量本身的存储类型。编译系统按指定的存储类型为结构指针 pman 和 pd 分配内存空间。

2.由于结构指针是指向整个结构体而不是某个成员，因此，结构指针的增（减）量运算，例如：pd++将跳过一个结构变量的整体，而不是只跳过其中的某个成员。其指针的增量值取决于它跳过的结构变量占用的内存。

3.结构体可以嵌套，即结构体成员又是一个结构变量或结构指针。

例如： struct student
 {
 char name[20];
 short age;
 char adr[30];
 struct Date BirthDay;
 struct Date StructDate;
 }stud[300];

其中：成员项 BirthDay、StudyDate 是具有 Date 结构类型的结构变量，称 student 为外层结构体，Date 为内层结构体，内层结构必须在外层之前定义。

4.使用结构指针结构成员进行引用时，有两种形式。

(1) 使用运算符“.”，其一般形式为

(*结构指针名).成员名

由于成员选择运算符“.”的优先级比取内容运算符“*”高，所以需要使用圆括号。

(2) 使用运算符“->”

结构指针名->成员名

例如：pman->old 在这种表示方法中，->也是一种运算符，它在运算符第一优先级中。它表示的意义是，访问指针指向的结构中的成员项。

下面举个例子简单分析一下

```
#include<stdio.h>
struct Date
{
    int month;
    int day;
    int year;
};
void main()
{
    struct Date today,*date_p;    //定义一个结构变量和结构指针变量
    date_p=&today;                //将结构指针指向一个结构变量
    date_p->month=4;               //采用结构指针给目标变量赋值
    date_p->day=15;
```

```
    date_p->year=1990;
}
```

9.3、结构在函数间的传递

9.3.1、结构体变量值传递

将一个结构体变量的值传递给另一个函数，有以下三种方法：

1.用结构体变量的成员作参数，用法和用普通变量作实参是一样的，属于值传递方式。这种方法在实际应用中很少使用。

2.用结构体变量作实参，把结构作为参数传递给函数，采用的是值传递的方式，将结构体变量所占的内存单元的内容全部顺序传递给形参。形参也必须是同类型的结构体变量。在函数调用期间，形参也要占用内存单元。

3.用结构变量的地址或结构数组的首地址作为实参，用指向相同结构类型的结构指针作为函数的形参来接受该地址值。用结构指针作为函数的形参与用指针变量作为形参在函数间传递方式是一样的，即采用地址传递方式，把结构变量的存储首地址或结构数组名作为实参向函数传递，函数的形参是指向相同结构类型的指针接收该地址值。相比于前两种，第三种方法在考试和实际中用的都是最多的，所以，应当重点掌握。

9.3.2、典型例题

1. 某班有若干名学生，每一名学生的信息包括学号、姓名、2 门课的成绩。

```
#include<stdio.h>
struct student          //定义学生结构体类型
{
    long int number;    //学号
    char name[8];       //姓名
    float score[2];     //2 门课成绩
};
void stu_output(struct student stud);
void main()
{
    struct student stu[3];
    int i,j;
    for(i=0;i<3;i++)
    {
        scanf("%ld",&stu[i].number);
        scanf("%s",stu[i].name);
        for(j=0;j<2;j++)
            scanf("%f",&stu[i].score[j]);
    }
    for(i=0;i<3;i++)
        stu_output(stu[i]);
}

//声明输出函数 stu_output
void stu_output(struct student stud)    //定义输出函数
{
    int j;
    printf("%ld",stud.number);
    printf("\t%s",stud.name);
```

- ```

 for(j=0;j<2;j++)
 printf("\t%.1f",stud.score[j]);
 putchar('\n'); }

```
2. 采用值传递方式在函数间传递结构变量，计算雇佣天数和工资

```

#include<stdio.h>
struct Date //定义结构
{
 int day; //定义天数
 int month; //定义月份
 int year; //定义年份
 int yearday;
 char mon_name[4];
};
int day_of_year(struct Date pd);
void main()
{
 struct Date HireDate;
 float laborage;
 scanf("%f",&laborage);
 scanf("%d",&HireDate.year);
 scanf("%d",&HireDate.day);
 HireDate.yearday=day_of_year(HireDate); //计算某年某月某日是这一年的第几天
}
int day_of_year(struct Date pd)
{
 int day_tab[2][13]={
 {0,31,28,31,30,31,30,31,31,30,31,30,31},//非闰年的每月天数
 {0,31,29,31,30,31,30,31,31,30,31,30,31}};//闰年的每月天数
 int i,day,leap; //leap 为 0 是非闰年，为 1 是闰年
 day=pd.day; //将每月的天数加入
 leap=(pd.year%4==0)&&(pd.year%100!=0)|| (pd.year%400==0);
 for(i=1;i<pd.month;i++) //求元月 1 日起累加天数
 day+=day_tab[leap][i];
 return day;
}

```

## \*9.4、位字段

### 9.4.1、位字段结构

位字段是一种数据压缩形式，数据整体没有具体意义。因此在处理位字段数据时，总是以组成它的位字段为处理对象。在 C 语言中，可以使用位操作（位逻辑与“&”操作、位逻辑或“|”操作等）对位字段进行处理。此外，C 语言还提供了处理位字段的另一种构造类型——位字段结构。位字段结构是一种特殊形式的结构，它的成员是二进制位字段。位字段结构定义中的每一个成员项一般书写格式是：

**unsigned 成员名: 位数;**

位字段以单个 unsigned 型数据为单位，其存放着一连串相邻的二进制位。

注：1.若某一位段要从另一个字开始存放，则可以使用下列形式：

```
struct test
{
 unsigned a:1;
 unsigned b:2; //成员 a 和 b 占用一个存储单元
 unsigned :0;
 unsigned c:3; //成员 c 占用另一个存储单元
};
```

2.在位字段结构体内若有不带名字的位字段时，冒号后面的位数应写 0，它具有特殊的含义，表示位字段间的填充物，用无名的位字段作为填充物占满整个 unsigned 型数据的其他位，使得下一个成员项 c 分配在相邻的下一个 unsigned 型。

3.位字段结构在程序中的处理方式和表示方法与普通结构相同。

4.一个位段必须存储在同一个存储单元中（不能跨两个单元）。如果一个单元空间不能容纳下一位段，则不用该空间，而从另一个单元起存放该位段。位段的长度不能大于存储单元的长度，也不能定义位段数组。

5.位字段结构的成员项可以和一个变量一样参加各种运算，但一般不能对位字段结构的成员项作取地址&运算。

## 9.5、联合

### 9.5.1、联合体的定义与使用

在 C 语言中，不同数据类型的数据可以使用共同的存储区域，这种数据构造类型称为联合体，简称联合。联合体在定义、说明和使用形式上与结构相似。**两者最本质的区别在于使用内存的方式上。**

联合的定义形式一般如下：

```
union 联合名
{
 数据类型 成员名 1;
 数据类型 成员名 2;

 数据类型 成员名 n;
};
```

虽然联合体和结构体的定义方式相同，但是联合体的使用内存方式却截然不同。

例如： union uarea

```
{
 char c_data;
 int i_data;
 long l_data;
};
```

宣布联合体由三个成员组成，这 3 个成员在内存中使用共同的存储空间。由于联合中各成员的数据长度往往不同，所以联合体在存储时总是按其成员中数据长度最长的成员项占用内存空间进行存储。如上述联合 uarea 占用内存的长度与成员项 l\_data 相同，即占用 4 字节内存。这 4 字节的内存位置上既存放 c\_data、又存放 i\_data，也存放 l\_data 的数据。联合一经定义，就可以定义使用这种数据构造类型的具体对象，即联合变量的定义，其形式与结构变量的定义类似。例如： union uare udata,\*pud,data[10];

由于联合体的各个成员项使用共同的内存区域，所以联合体的内存空间中在某个时刻只能保持某个成员项的数据。由此可知，在程序中参加运算的必然是联合体的某个成员项。联合体成员项的表现形式与结构相同，它们也使用访问成员运算符.或->表示。

在程序中经常使用结构体和联合体相互嵌套的形式，例如：

```
union uarea
{
 char c_data;
 int i_data;
 long l_data;
};
struct data
{
 char *p;
 int type;
 union uarea udata;
};
```

## 9.6、类型定义语句 typedef

### 9.6.1、用 typedef 语句定义新类型名

C 语言为编程者提供了一种新的类型名来代替已有的基本数据类型名和已经定义了的类型。其中，typedef 是类型定义语句的关键字，类型说明是对新类型名的描述，类型说明可以是各种基本数据类型名（char, int, ……，double）和已经定义了的结构体、联合体、指针、数组、枚举。

例如：      typedef int INTEGER;  
              typedef float REAL;

即 int 与 INTEGER, float 与 REAL 等价，以后在程序中可任意使用两种类型去说明具体的变量的数据类型。

例如：      int i, j;      等价      INTEGER i, j;  
              float a, b    等价      REAL a, b;

在程序中合理使用 typedef 可以提升程序的可读性。[习惯上把新类型名用大写字母来表示。](#)

### 9.6.2、新类型名的应用

[用 typedef 只是对已经存在的类型名增加了一个新的类型名，并没有创建一个新的数据类型。](#)

typedef 语句和#define 语句的区别：

```
typedef int COUNT;
#define COUNT int
```

上述两条语句的作用都是给 int 起一个新的变量名叫 COUNT，但是#define 语句是在预处理时进行字符串的替换的，而 typedef 是在编译时进行处理的，它并不是进行简单的字符串的替换。例如，定义字符数组类型时，若写

```
typedef char[81] STRING;
```

则定义了一个 STRING 类型。它是具有 81 个字符的数组，以后就可用 STRING 类型定义类同的字符数组。

若写： STRING text, line;

使用 typedef 语句给已定义的结构类型赋予新的类型名，大大简化了对结构变量的说明。

```
例如： typedef struct
{
 double real;
 double imag;
}COMPLEX;
```

其中：COMPLEX 就是 struct{...}的新类型名，即 struct{...}与 COMPLEX 对其结构变量的说明作用一样，显然 COMPLEX 也是结构名。

例如：      COMPLEX c1, c2;

```
COMPLEX *r,*op1,*op2;
```

但是 COMPLEX 是结构类型，即是形式结构而不是结构变量，因此**不能用 COMPLEX 进行访问成员的运算**。如“COMPLEX.real”是非法的。

下面，举个例子分析以下

求复数的加法运算的程序。

```
#include<stdio.h>
typedef struct
{
 double real;
 double imag;
}COMPLEX; //COMPLEX 就是 struct{...}的新类型名
//复数的加法运算函数，两个形参都是指向 COMPLEX 类型的结构指针，其返回值仍是一个复数结构类型
COMPLEX caadd(COMPLEX *op1,COMPLEX *op2);
COMPLEX caadd(COMPLEX *op1,COMPLEX *op2)
{
 COMPLEX result;
 result.real=op1->real+op2->real; //实数部分相加
 result.imag=op1->imag+op2->imag; //虚数部分相加
 return result;
}
void main()
{
 COMPLEX a={6.0,8.0},b={2.0,8.0},c;
 printf(" (1) COMPLEX a(%2lf,%2lf)\n",a.real,a.imag);
 printf(" (2) COMPLEX b(%2lf,%2lf)\n",b.real,b.imag);
 c=cadd(&a,&b); //调用 cadd()求复数 a 和 b 之和
 printf(" (3) COMPLEX c(%2lf,%2lf)\n",c.real,c.imag);
}
```

该程序的输出结果为

```
(1) COMPLEX a(6.00,8.00)
(2) COMPLEX b(2.00,8.00)
(3) COMPLEX c(8.00,16.00)
```

## 9.7、用指针处理链表

### 9.7.1、建立简单的静态链表

1.链表是一种常见的重要的数据结构。它是存储分配的一种结构。

2.链表有一个“头指针”变量，一般以 head 表示，**它没有数据域，只有指针域**。它存放一个地址，该地址指向第一个元素。链表中每一个元素称为“节点”，每一个节点包含两个部分：**(1) 用户需要用的实际数据；(2) 下一个节点的地址指向下一个节点的指针**。

3.链表中各元素在内存中的地址是不连续的。也就是说，想要找到一个元素，即找到其中一个节点，必须找到它的上一个节点。**如果不提供“头指针”，则整个链表都无法访问**。

前面介绍了结构体变量，用它去建立链表是最合适的。一个结构体变量包含若干成员，这些成员可以是数值类型、字符类型、数组类型，也可以是指针类型。用指针类型成员来存放下一个节点的地址。

例如：

```
struct Student
{
```

```

int num;
float score;
struct Student *next; //next 是指针变量，指向结构体变量
};

```

其中成员 `num` 和 `score` 用来存放节点中的有用数据（用户需要用到的数据），`next` 是指针类型的成员，它指向 `struct Student` 类型数据（就是 `next` 所在的结构类型）。一个指针类型的成员既可以指向其他类型的结构体数据，也可以指向自己所在的结构体类型的数据。现在，`next` 是 `struct Student` 类型的数据。用这种方法就可以建立链表。

下面，举一个简单的例子来说明怎样建立和输出一个链表

```

#include<stdio.h>
struct Student //声明结构体类型 struct Student
{
 int num;
 float score;
 struct Student *next;
};
int main()
{
 struct Student a,b,c,*head,*p; //定义 3 个结构体变量 a, b, c 作为链表的节点
 a.num=10101;a.score=89.5; //对三个节点添加内容
 b.num=10103;b.score=90;
 c.num=10107;c.score=85;
 head=&a; //将节点 a 的起始地址赋给头指针 head
 a.next=&b; //将 b 的起始地址赋给 a 节点的 next 变量
 c.next=NULL; //c 节点的起始地址赋给 a 节点的 next 成员
 p=head; //使 p 也指向 a 节点
 do
 {
 printf("%ld%5.1f\n",p->num,p->score); //输出 p 指向的节点数据
 p=p->next; //使 p 指向下一个节点
 }while(p!=NULL); //输出完 c 节点后 p 的值为 NULL，循环终止
 return 0;
}

```

输出结果为:10101 89.5  
10103 90.0

### 9.7.2、建立动态链表与链表的输出

在学习动态链表之前，先来了解几个函数

#### 1.malloc 函数

`void *malloc(long NumBytes);` 该函数分配了 `NumBytes` 个字节，并返回了指向这块内存的指针。如果分配失败，则返回一个空指针（`NULL`）。

#### 2. calloc 函数

`void *calloc(unsigned n,unsigned size);` 在内存的动态存储区中分配 `n` 个长度为 `size` 的连续空间，函数返回一个指向分配起始地址的指针；如果分配不成功，返回 `NULL`。

#### 3.free 函数

`void free(void *p);` 释放指针 `p` 所指向的内存区，无返回值。

下面通过一道例题来分析以下：

写一函数建立一个有 3 名学生数据的单向动态链表。

```
#include<stdio.h>
#include<stdlib.h>
#define LEN sizeof(struct Student)
struct Student
{
 long num;
 float score;
 struct Student *next;
};
int n; //n 为全局变量，本文件模块中各函数均可用它
struct Student *creat(void) //定义函数。此函数返回链表头节点
{
 struct Student *head;
 struct Student *p1,*p2;
 n=0;
 p1=p2=(struct Student *)malloc(LEN); //开辟一个新单元
 scanf("%ld,%f",&p1->num,&p1->score); //输入第 1 各学生的学号和成绩
 head=NULL;
 while(p1->num!=0)
 {
 n=n+1;
 if(n==1)head=p1;
 else p2->next=p1;
 p2=p1;
 p1=(struct Student *)malloc(LEN); //开辟动态存储区，把起始地址赋给 p1
 scanf("%ld,%f",&p1->num,&p1->score); //输入其他学生的学号和成绩
 }
 p2->next=NULL;
 return(head);
}
int main()
{
 struct Student *pt;
 pt=creat(); //函数返回链表第一个节点的地址
 printf("\n%ld\n%5.1f\n",pt->num,pt->score); //输出第一个节点的成员值
 return 0;
}
当需要输出链表时:
.....
void printf(struct Student *head)
{
 struct Student *p;
 p=head;
 if(head!=NULL)
do
{
```

```

 printf("%ld%5.1f\n",p->num,p->score);
 p=p->next;
 }while(p!=NULL);
}

```

说明：链表是一个比较深入的内容，初学者有一些难度，并且在大多数的 C 语言考试中不会作为考试的重点，只需要对此有一定了解即可，在以后用到时再进一步学习。

## 【解题技巧】

**例 9.1** 设有定义：

```

struct complex
{ int real ,unreal;}data1={1,8},data2;

```

则以下赋值语句中错误的是 ( )

A) data1=data2;

B) data2={2,6};

C) data2.real=data1.real;

D) data2.real=data1.unreal;

**正解：**B

**分析：**如果想要给结构变量 data2 直接赋值，应该写成：struct complex data2={2,6};

**例 9.2** 已知学生记录描述为：

```

struct student
{
 int no;
 char name[20];
 char sex;
 struct
 { int year;
 int month;
 int day;
 } birth;
};

```

struct student s;

设变量 s 中的“生日”应是“1995 年 1 月 16 日”，下列对“生日”的正确赋值方式是 ( )

- A) year=1995; month=1; day=16;
- B) birth.year=1995; birth.month=1; birth.day=16;
- C) s. birth.year=1995; s. birth.month=1; s. birth.day=16;
- D) s.year=1995; s.month=1; s.day=16;

**正解：**C

**分析：**本题是考察结构的嵌套定义，一般形式为 第一级结构名. 第二级结构名. .... 第 n 级结构名. 成员名。故本题选 C。

**例 9.3** 在说明一个结构体变量时，系统分配给它的存储空间是( )

- A) 该结构体中第一个成员变量所需存储空间
- B) 该结构体中最后一个成员变量所需存储空间

- C)该结构体中占用最大存储空间的成员变量所需存储空间
- D)该结构体中所有成员变量所需存储空间的总和

**正解:** D

**分析:** 结构体在定义时,系统会自动为它的所有变量分配空间,因此它的存储空间是结构体中所有成员变量所需要的存储空间,故选 D

**例 9.4** 在说明一个联合体变量时,系统分配给它的存储空间是( )

- A)该联合体中第一个成员变量所需存储空间
- B)该联合体中最后一个成员变量所需存储空间
- C)该联合体中占用最大存储空间的成员变量所需存储空间
- D)该联合体中所有成员变量所需存储空间的总和

**正解:** C

**分析:** 由于联合体中各个成员的数据长度往往不同,因此系统在存储时总是按其成员中数据最大的成员项分配空间。

## 【精选习题】

答案 P124

### 基础篇

1. 设有如下定义:

```
struct sk
{
 int a;
 float b;
}data;
int *p;
```

若要使 p 指向 data 中的 a 成员,正确的赋值语句是( )

- A) p=&a;
- B) p=data.a;
- C) p=&data.a;
- D) \*p=data.a;

2. 有以下程序

```
#include<stdio.h>
struct tt
{
 int x;
 struct tt *y;
}*p;
struct tt a[4]={20,a+1,15,a+2,30,a+3,17,a};
void main()
{
 int i;
 p=a;
 for(i=1;i<=2;i++)
 {
 printf("%d",p->x);
 p=p->y;
```

```

 }
}

```

程序运行的结果是 ( )

- A)20, 30,                      B)30, 17,                      C)15, 30,                      D)20, 15,

3.设有语句

```

struct st
{
 int n;
 int *t; } *p;
static struct st s[3]={5,'\0',7,'\0',9,'\0'};
p=s;

```

则 printf("%d\n",\_\_\_\_\_) 的值是 6。

- A) p++->n                      B) p->n++                      C) (\*p).n++                      D) ++p->n

4.有以下程序

```

#include<stdio.h>
struct ord
{
 int x,y;
} dt[2]={1,2,3,4};
void main()
{
 struct ord *p=dt;
 printf("%d,",++(p->x));
 printf("%d\n",++(p->y));
}

```

程序运行的结果为 ( )

- A)3,4                      B)4,1                      C)2,3                      D)1,2

5.若有以下说明和语句, 已知 char 型占 1 个字节, int 类型占 2 个字节, double 型占 8 个字节, 则下面程序段的输出结果为 ( )

```

struct st
{
 char a[10];
 int b;
 double c; };
printf("%d\n",sizeof(struct st));

```

- A)10                      B)20                      C)8                      D)28

6.以下程序运行的结果为 ( )

```

#include<stdio.h>
#include<string.h>
struct A
{
 int a;
 char b[10];
 double c;
};

```

```

void f(struct A t);
void main()
{
 struct A a={1001,"ZhangDa",1098.0};
 f(a);
 printf("%d,%s,%6.1f\n",a.a,a.b,a.c);
}
void f(struct A t)
{
 t.a=1002;
 strcpy(t.b,"ChangRong");
 t.c=1202.0;
}

```

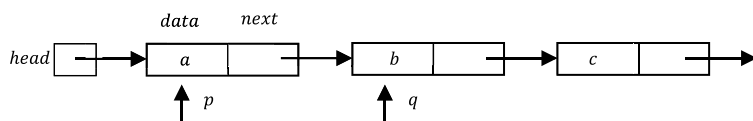
- A)1002,ZhangDa, 1202.0  
 B)1002,ChangRong,1202.0  
 C) 1001,ChangRong,1098.0  
 D)1001,ZhangDa,1098.0

## 提高篇

- 以下叙述中错误的是 ( )
  - A)可以用 typedef 将已存在的类型用一个新的名字来代表
  - B)可以通过 typedef 增加新的类型
  - C)用 typedef 定义新的类型名后, 原有的类型名仍有效
  - D)用 typedef 可以为各种类型起别名, 但不能为变量起别名
- 以下叙述中正确的是 ( )
  - A)使用 typedef 说明类型名时, 其格式是: typedef 新类型名 原类型名;
  - B)在程序中, 允许 typedef 来说明一种新的类型名
  - C)使用 typedef 说明新类型时, 后面不能加分号
  - D)在使用 typedef 改变原类型的名称后, 只能使用新的类型名
- 若有定义
 

```
typedef int *T; T a[10];
```

 则 a 的定义与下面哪个语句等价 ( )
  - A)int (\*a)[10];
  - B)int \*a[10];
  - C)int \*a;
  - D)int a[10];
- 假设已建立以下链表结构, 且指针 p 和 q 已指向如图所示的节点:



- 则以下选项中可将 q 所指节点从链表中删除并释放该节点的语句组是 ( )
- A)p->next=q->next;free(q);
  - B)p=q->next;free(q);
  - C)p=q;free(q);

D)(\*p).next=(\*q).next;free(p);

5.有以下程序段

int \*p;

p=\_\_\_\_\_malloc(sizeof(int));

若要求使 p 指向一个 int 型的动态存储单元，在横线处应填入的是 ( )

A) (int \*)

B)int

C)int \*

D) (\*int)

6.平面上的一个点的位置可以用其横坐标和纵坐标确定：

(1) 请根据结构体 struct POINT 类型的定义来描述平面上的一个点；

(2) 根据以下函数的声明编写函数，判断一个点是否在单位圆内，如果在，则返回 1，否则返回 0；

int in\_circle(struct POINT);

7.设有 N 名考生，N 由宏定义设定，每个考生的数据包括考生号、姓名、性别和成绩。编程实现考生所有信息的输入，按成绩由高到低对考生进行排序，并统计考生的及格率，最后输出排序后的考生数据和及格率。要求考生信息采用结构体类型表示，考生排序和及格率计算分别采用函数实现，及格线为 60 分。

8.一个公司，有若干名员工，每名员工有姓名，性别，工龄，工资等信息。编程输入并建立员工档案信息和便于工资发放的各种钞票数（工资为整数，发放的工资各种钞票限定为 100 元，50 元，20 元，10 元，5 元，1 元，发放的钞票张数要求为最少），要求输出工龄大于 20 年，工资高于 5000 元的所有男员工信息。（要求输入和输出功能用不同的函数实现，编写主函数完成上述函数的调用）