C语言基础Day7-结构体

一、结构体概述

将多个相同或者不同类型的数据存放在一块连续的内存中

二、结构体定义与初始化

结构体模型和结构体变量的关系：

- 结构体类型：指定一个结构体类型，他相当于一个模型，但是其中并没有数据，系统对其也不分配实际的内存单元
- 结构体变量：系统根据结构体类型（内部成员状况）为之分配空间

两种操作结构体成员的方法：

- 通过点域
- 通过结构体地址

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

// 关键字 struct 代表这个是一个结构体类型
// stu 代表结构体的名字
// 整个结构体的类型是 struct stu
// 结构体类型struct stu{} 中是结构体的成员 一个有三个成员，每一个成员的类型可以是任意的类型
// 注意：定义结构体的时候 struct stu只是一个类型  一个模板 没有内存空间 不可以给结构体成员赋值

struct stu
{
    int id;
    int age;
    char name[128];


}a,b;// 需要加上分号  定义类型时 同时定义了两个结构体变量 同struct stu a,b

int main()
{
    // struct stu d = {1,2,"hello"};// 结构体定义变量 开辟内存空间
    // struct stu e = {.age = 20};// 给部分成员初始化  其他成员内容为0

    // 通过结构体变量操作结构体成员  使用点域.操作
    struct stu d;
    d.id = 2;
    d.age = 20;
```

```
    strcpy(d.name,"hello");// 需要拷贝 不可以直接赋值
    printf("%d %d %s\n",d.id,d.age,d.name);

    // 通过结构体地址操作结构体成员->
    (&d)->id = 3;
    (&d)->age = 20;
    strcpy((&d)->name,"world");

    printf("%d %d %s\n",(&d)->id,(&d)->age,(&d)->name);// 通过取地址 操作结构体成员

    return 0;
}
```

三、结构体数组

一个数组，数组的每一个元素都是一个结构体

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

// 关键字 struct 代表这个是一个结构体类型
// stu 代表结构体的名字
// 整个结构体的类型是 struct stu

// 结构体类型struct stu{} 中是结构体的成员 一个有三个成员，每一个成员的类型可以是任意的类型
// 注意：定义结构体的时候 struct stu只是一个类型  一个模板 没有内存空间 不可以给结构体成员赋值

struct stu
{
    int id;
    int age;
    char name[128];


}a,b;// 需要加上分号  定义类型时 同时定义了两个结构体变量 同struct stu a,b

// int main()
// {
//     // struct stu d = {1,2,"hello"};// 结构体定义变量 开辟内存空间
//     // struct stu e = {.age = 20};// 给部分成员初始化  其他成员内容为0

//     // 通过结构体变量操作结构体成员  使用点域.操作
//     struct stu d;
//     d.id = 2;
//     d.age = 20;
//     strcpy(d.name,"hello");// 需要拷贝 不可以直接赋值
//     printf("%d %d %s\n",d.id,d.age,d.name);
```

```
//     // 通过结构体地址操作结构体成员->
//     (&d)->id = 3;
//     (&d)->age = 20;
//     strcpy((&d)->name,"world");

//     printf("%d %d %s\n",(&d)->id,(&d)->age,(&d)->name);// 通过取地址 操作结构体成员

//     return 0;
// }

int main()
{
    // 定义一个结构体数组  结构体数组有五个元素 每一个元素都是struct stu类型
    struct stu num[5] = {{1,20,"lucy"},{2,21,"bub"},{3,22,"peter"},{4,22,"maker"},{5,26,"ubuntu"}};

    // 打印结构体的元素
    for(int i = 0; i < sizeof(num) / sizeof(num[0]); i++)
    {
        printf("%d %d %s\n",num[i].id,num[i].age,num[i].name);
    }

    return 0;
}
```

四、结构体套结构体

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

// 关键字 struct 代表这个是一个结构体类型
// stu 代表结构体的名字
// 整个结构体的类型是 struct stu

// 结构体类型struct stu{} 中是结构体的成员 一个有三个成员，每一个成员的类型可以是任意的类型
// 注意：定义结构体的时候 struct stu只是一个类型  一个模板 没有内存空间 不可以给结构体成员赋值

struct stu
{
    int id;
    int age;
    char name[128];
}a,b;// 需要加上分号  定义类型时 同时定义了两个结构体变量 同struct stu a,b
```

```c
struct heima_stu
{
    /* data */
    struct stu s;// 嵌套结构体
    char subject[128];
};


int main()
{
    // // 定义一个结构体数组  结构体数组有五个元素 每一个元素都是struct stu类型
    // struct stu num[5] = {{1,20,"lucy"},{2,21,"bub"},{3,22,"peter"},
{4,22,"maker"},{5,26,"ubuntu"}};

    // // 打印结构体的元素
    // for(int i = 0; i < sizeof(num) / sizeof(num[0]); i++)
    // {
    //     printf("%d %d %s\n",num[i].id,num[i].age,num[i].name);
    // }

    struct heima_stu xxf;

    xxf.s.id = 1;
    xxf.s.age = 22;
    strcpy(xxf.s.name,"hello");
    strcpy(xxf.subject,"C++");

    printf("%d %d %s %s\n",xxf.s.id,xxf.s.age,xxf.s.name,xxf.subject);

    return 0;
}
```

五、结构体的赋值

同普通的变量一样，**结构体的赋值，也是需要传地址，如果赋值函数的形参只是一个结构体变量，那么改变的只是拷贝过来的变量，所以需要传地址，这样改变的就是原来的结构体变量的内容**

下面的代码就是错误的！

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define _CRT_SECURE_NO_WARNINGS

struct stu
{
    int id;
    int age;
    char name[128];
};
```

```
void memcpy_str(struct stu x,struct stu y)
{
    memcpy(&x,&y,sizeof(x));// 内存内容复制函数  第三个参数是 复制的字节大小
}

int main()
{
    struct stu a = {1,3,"nnn"};
    struct stu b;

    // 赋值内容
    memcpy_str(b,a);// 传入的参数是变量  不是变量地址  那么调用函数的时候只是复制一份变量 没有改变原来的变量内容
    printf("%d %d %s\n",b.id,b.age,b.name);
    return 0;
}
```

通过打印结果可以看到 b中没有任何内容，y中是有内容的，y中的内容是通过x复制得到的

传入参数是结构体变量的地址

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

struct stu
{
    int id;
    int age;
    char name[128];
};

void memcpy_str(struct stu *x,struct stu *y)
{
    memcpy(x,y,sizeof(*x));// 内存内容复制函数  第三个参数是 复制的字节大小
}

int main()
{
    struct stu a = {1,3,"nnn"};
    struct stu b;

    // 赋值内容
    memcpy_str(&b,&a);
    printf("%d %d %s\n",b.id,b.age,b.name);
    return 0;
}
```

当然，可以通过一个最直接的方法，直接赋值：b = a(相同类型的结构体变量，成员也要相同)

六、结构体指针

定义结构体指针需要开辟内存空间，没有开辟空间是不可以操作

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

struct stu
{
    int id;
    int age;
    char name[128];
};


int main()
{
    struct stu a;
    struct stu *p = &a;// p没有初始化的话 是一个野指针  随机指向内存

    // 或者申请堆区空间
    struct stu *q = (struct stu *)malloc(sizeof(struct stu));// 申请一块
sizeof(struct stu)大小的内存空间 然后q指针指向它

    p->id = 1;
    p->age = 22;
    strcpy(p->name,"ubuntu");// 赋值

    q->id = 10;
    q->age = 11;
    strcpy(q->name,"uuuuuu");

    printf("%d %d %s\n",p->id,p->age,p->name);
    printf("%d %d %s\n",q->id,q->age,q->name);
    return 0;
}
```

七、结构体套结构体指针（结构体成员是一个指针）

结构体中的指针变量 直接赋值字符串地址

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

```c
struct tea
{
    /* data */
    int id;
    char *p;
};


int main()
{
    struct tea *tmp = (struct tea *)malloc(sizeof(struct tea));

    tmp->id = 100;
    tmp->p = "hello";// 可以的 复制的是地址 hello在文字常量区的地址 不是直接拷贝的内容
    printf("%d %s\n",tmp->id,tmp->p);

    return 0;
}
```

结构体中的字符串指针，先开辟空间，再拷贝内容

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define _CRT_SECURE_NO_WARNINGS

struct tea
{
    /* data */
    int id;
    char *p;
};


int main()
{
    struct tea *tmp = (struct tea *)malloc(sizeof(struct tea));

    tmp->id = 100;
    // tmp->p = "hello";// 可以的 复制的是地址 hello在文字常量区的地址
    tmp->p = (char *)malloc(128);// 先开辟空间 在使用strcpy拷贝内容  否则是野指针
    strcpy(tmp->p,"hello");// 拷贝的是内容 不是地址
    printf("%d %s\n",tmp->id,tmp->p);

    return 0;
}
```

上面的代码类似于这样：

```
// 这样写是可以的 p存放hello的地址 即使p没有开辟堆区空间  但是p指向hello开辟的内存空间 p存放地址
char *p;
p = "hello";

char *p;
strcpy(p,"hello");// 这样写就是错误的  p没有开辟内存空间  所以不可以直接被赋值


char *p;
p = (char *)malloc(sizeof(char) * 128);
strcpy(p,"hello");// 这样写就是可以的
```

p直接赋值地址是没有问题的，拷贝内容的话必须要有内存空间才可以

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

struct t
{
    int a;
};

struct tea
{
    /* data */
    int id;
    char *p;
    struct t *b;
};

int main()
{
    struct tea *tmp = (struct tea *)malloc(sizeof(struct tea));

    tmp->id = 100;// 可以直接赋值
    tmp->p = (char *)malloc(100);// 需要开辟一块空间 在进行赋值 否则是野指针
    strcpy(tmp->p,"kkkk");


    // 下面的写法是错误的
    // tmp->b->a = 100;// 因为tmp->b是地址 所以使用->来访问a
    // 因为tmp->b 是指针  没有开辟堆空间 b是野指针 不能直接赋值b所指向的空间
    tmp->b = (struct t *)malloc(sizeof(struct t));
```

```
    tmp->b->a = 1000;

    free(tmp->p);
    free(tmp->b);
    free(tmp);// 先释放里面的内存  再释放外面的内存
    return 0;
}
```

八、结构体作为函数参数

8.1 结构体普通变量作为函数参数

值传递 并没有改变结构体的成员

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

struct stu
{
    char name[50];
    int age;
};

// 函数参数是结构体普通变量
void set_stu(struct stu tmp)
{
    strcpy(tmp.name,"mike");
    tmp.age = 10l;
}

int main()
{
    struct  stu s = {0};

    set_stu(s);// 值传递 没有进行初始化
    printf("%s %d\n",s.name,s.age);

    return 0;
}
```

8.2 结构体指针变量作为函数参数-地址传递

地址传递，函数中通过指针来操纵结构体变量的内存空间

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

```c
#include<string.h>
#define _CRT_SECURE_NO_WARNINGS

struct stu
{
    char name[50];
    int age;
};

// 函数参数是结构体普通变量
void set_stu(struct stu *tmp)
{
    strcpy(tmp->name,"mike");
    tmp->age = 10l;
}

int main()
{
    struct  stu s = {0};

    set_stu(&s);// 地址传递  函数中通过指针 操作s的内存空间
    printf("%s %d\n",s.name,s.age);

    return 0;
}
```

8.3 结构体数组名作为函数参数

```c
#include<stdio.h>
#include<stdlib.h>
#define _CRT_SECURE_NO_WARNINGS
#include<string.h>

struct c13
{
    int id;
    char name[128];
    /* data */
};

// 传入数组名 其实就是传入地址
void set_num(struct c13 *p,int n)
{
    for(int i = 0; i < n; i++)
    {
    //    (*(p + i)).id = i + 10;
    // 或者写成
        p[i].id = 10 + i;

        char buf[128] = "";// 初始化一个空的字符串
```

```
        sprintf(buf,"%d%d%d",i,i,i);

        strcpy(p[i].name,buf);//拷贝字符串

    }
}

int main()
{
    struct c13 num[3];

    // 结构体内容清0
    memset(num,0,sizeof(num));// 所有字节内容都清0

    set_num(num,sizeof(num) / sizeof(num[0]));// 数组名作为参数传入函数 其实就是传入数组地址  指针类型

    for(int i = 0; i < sizeof(num) / sizeof(num[0]); i++)
    {
        printf("%d %s\n",num[i].id,num[i].name);
    }


    return 0;
}
```

8.4 const修饰结构体指针

```
struct c a;
struct c b;

const struct c *p = &a;// const修饰的是*p 不能通过指针p修改P指向的那块内存空间
p->id = 100;// 错误的代码


struct c *const p = &a;
p = &b;// const修饰的是指针变量p 不是*p 所以不可以修改p指向的内存地址  错误的代码
```