step further

专题

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与联系（模块设计方法、函数）

表达与转换（基本操作、数据类型）

提高：

构造与访问（数组、指针、**结构体**）

归纳与推广（程序设计的本质）

- **结构体的基本概念**
  - 结构类型的构造
  - 结构体的定义与初始化
  - 结构体的操作
    - **含有指针成员的结构体及其操作**
- **用指针操纵结构体**
  - **用指针操纵含有指针成员的结构体**

问题的提出

- 描述一位学生的信息：

num	name	F/M	age	score	addr
191220999	Hans	M	19	395	Nanjing

- 数组？

（数组类型用于表示 固定多个 同类型 的数据群体）

结构(struct)类型

属性（元素）

- C语言的结构类型用于表示由固定多个 类型可以不同 的成员所构成的数据群体。
  - 固定多个
  - 类型可以不同的数据群体（含义可以不同的相关信息）
  - 成员间在逻辑上没有先后次序关系，其说明次序仅影响成员的存储安排，不影响操作，成员有成员名
  - 相当于其他高级语言中的“记录”
- 数组类型：
  - 固定多个
  - 同类型（相同意义的相关信息）
  - 元素间在逻辑上有先后次序关系，按序连续存储，元素有下标

结构类型的构造

```
struct Student
{
   int number;    //成员
   char name;     //成员
   int age;       //成员
};

typedef struct
{
   int number;
   char name;
   int age;
}Student;
```

一种 tag

宣布组成的
成员名称和成员类型

```
struct Date
{
   int month;
   int day;
   int year;
};
```

注意：构造结构类型时，
花括号中至少要定义一个成员。
除void类型和本结构类型外，
结构成员可以是其他任意的类型。

- 结构类型标识符与其成员或其他变量可重名，
- 结构类型成员和其他变量也可以重名，
- 同一个结构体的各个成员不能重名。

- 即使两个结构类型中的成员类型、名称、顺序都完全一致，它们也是不同的结构类型。不同结构类型的成员可以重名，不过结构类型标识符不能重名。

- 如果在函数内部构造结构类型，则该函数之外此结构类型不可用。一般把结构类型的构造放在文件头部，也可以把结构类型的构造放在头文件中。

结构体的定义

- 可以用构造好的结构类型来定义结构体。
  - 比如，

```
struct Student s1, s2;
struct Date d1, d2;
```

```
typedef struct
{     …;
}Student;
```

  - 结构体定义中，前面的struct是否可以省略跟开发环境有关，如果用之前typedef形式指定的类型名，则定义结构体时不用加struct

- 也可以在构造结构类型的同时直接定义结构体，
  - 比如，

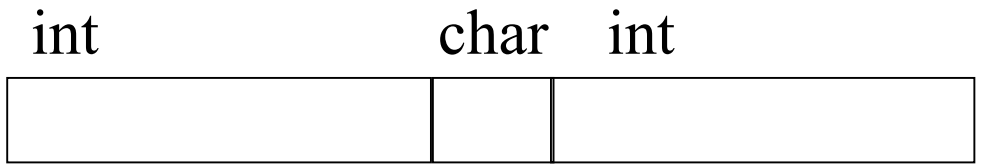
```
struct Student
{       int number;
        char name;
        int age;
} s1, s2;
```

```
struct Employee
{
  int number;
  char name;
  struct Date
  {
     int   year;
     int   month;
     int   day;
  } birthday ; //其他类型的结构体可以作为本结构类型的成员
} e ;
```

```
struct Date
{   int   year;
    int   month;
    int   day;
};
struct Employee
{   int number;
    char name;
    Date birthay;
} e ;
```

- 系统按构造时的顺序为各个成员分配空间

```
struct Student
{
   int number;
   char name;
   int age;
} s ;
```



（a）

- 系统往往以字为单位给结构体分配空间

（b）

（c）

- 结构体一般不加register修饰

结构体的初始化

- 可以在定义结构体的同时，给各个成员赋值，即结构体的初始化。
  - 比如，

```
Student s1, s2 = {1220001, 'T', 19};
Employee e = {1160007, 'J', {1996, 12, 26}};
```

  - 注意：在构造一个结构类型时，不能对其成员进行初始化，因为构造类型时，编译器不分配存储空间。比如，

```
struct Student
{
 int number = 1220001;      //此处的初始化是错误的
 char name;
 int age;
};
```

C++11新标准允许给一个默认值

const

```
struct
{  char  name[20];
   struct Date
   {     int   year;
         int   month;
         int   day;
    }birthday;
} s={"Joe", {1996,12,14}};
```

```
s.name = "Joe";×
```

```
scanf("%s", s.name);
```

```
struct
{  char  *name;
   struct Date
   {     int   year;
         int   month;
         int   day;
    }birthday;
} s={"Joe", {1996,12,14}};
```

```
s.name = "Joe";
```

```
scanf("%s", s.name);×
```

结构体的操作

- 对结构体的操作常常是通过成员操作符操作结构体的成员完成的。访问成员的格式为：
  - <结构体名> . <成员名>

    ```
    s2.age = 19;
    ```

  - 点号是成员操作符，它是双目操作符，具有1级优先级，结合性为自左向右。
  - 由于对成员的访问不是按次序，而是按名称访问，所以，构造结构类型时成员的排列顺序无关紧要。

- 如果某成员类型是另一个结构类型，则可以用若干个成员操作符访问最低一级的成员。比如，

```
e.birthday.year = 1996;
```

```
struct Employee
{
  int number;
  char name;
  struct Date
  {
     int   year;
     int   month;
     int   day;
  } birthday ;
} e ;
```

- 结构类型可以含有**指针成员**，对指针成员的操作方法与其他类型成员类似。比如，

```
struct
{
  int no;
  int *p;
} s ;

s.no = 1001;
s.p = &s.no;
```

赋值操作

- 相同结构类型的不同结构体之间可以直接相互赋值，其实质是两个结构体相应的存储空间中的所有成员数据直接拷贝。比如，

```
Employee e1, e2;
e1 = e2;
```

  - 或者，

```
typedef Employee Employ
Employee e1;
Employ e2;
e1 = e2;
```

- 不同结构类型的结构体之间不能相互赋值，

下面a、b两个结构体不可以相互赋值：

```
struct
{     int no;
      char name;
} a ;

struct
{     int no;
      char name;
} b ;
a = b;  ×
```

结构体作为函数参数/返回值

- 可作为参数传给函数：默认参数传递方式为值传递

  （实参和形参都是结构体名，类型相同；但实参和形参代表两个不同的结构体，运行时分配不同的存储空间。）

- 函数也可以返回一个

  结构体（结构类型函数）

例8.1 验证结构体的值传递方式。

```
void myFun(Stu s1)
{ s1.name = 'J';
  s1.score = 100.0;
  printf("%c: %f\n", s1.name, s1.score);
}

int main( )
{ struct Stu stu1;
  stu1.name = 'T';
  stu1.score = 90.0;
  printf("%c: %f\n", stu1.name, stu1.score);
  myFun(stu1);
  printf("%c: %f\n", stu1.name, stu1.score);
  return 0;
}
```

```
enum FeMale {F, M};
struct Stu
{
  int id;
  char name;
  FeMale s;
  int age;
  float score;
};
```

程序结果会显示:

```
T: 90.0
J: 100.0
T: 90.0
```

用指针操纵结构体

重点

- 将某结构体的地址赋给基类型为该结构类型的指针变量，则可以用这个指针变量操纵该结构体的成员，这时成员操作符写成箭头形式（->），而不是点形式（.）。比如，

```
struct
{
  int no;
  float score;
} s, *ps ;
ps = &s;
ps -> no = 1001;          //相当于(*ps).no或s.no
ps -> score = 90.0;       //相当于(*ps).score或s.score
```

重点

- **如果有成员是另一结构类型的指针变量，则可以用若干个箭头形式的成员操作符访问最低一级的成员。比如，**

```
struct
{
  Student *p1;
  float score;
} s, *pps ;
pps = &s;
pps -> p1 = &s1;
pps -> score = 85;
pps -> p1 -> number = 1220001;
pps -> p1 -> name = 'Q';
pps -> p1 -> age = 19;
```

```
struct Student
{
      int number;      //成员
      char name;       //成员
      int age;   //成员
} s1;
```

重点

- 为了提高程序的效率，函数间传递结构体时，实参可以用结构体的地址，形参用相同结构类型的指针。

传值方式-效率不高

- 例8.2-1 未用指针变量操纵结构体。

```
int main( )
{
    struct Date d1;
    scanf("%d%d%d", &d1.year, &d1.month, &d1.day);
    Days(d1);

    return 0;
}
```

d1


```
void Days(struct Date d2)
```

d2


```
struct Date
{
    int year;
    int month;
    int day;
    int yearday;
};
```

```
void Days(struct Date d2)
{
    int monthtable[ ][13]= {
        {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
        {1, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

    int i, leap = 0;
    d2.yearday = d2.day;

    if( ((d2.year %4 == 0) && (d2.year %100 != 0))
                          || (d2.year % 400 == 0) )
        leap = 1;

    for(i = 1; i < d2.month; ++i)
        d2.yearday += monthtable[leap][i];
    printf("所输入的日期是该年的第几天: %d", d2.yearday);
}
```

传址方式-提高效率

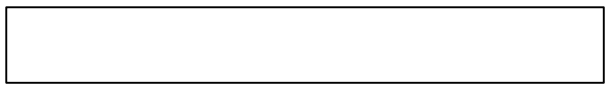
- 例8.2-2 用指针变量操纵结构体。

d1


```
int main( )
{
    struct Date d1;
    scanf("%d%d%d", &d1.year, &d1.month, &d1.day);
    Days(&d1);

    return 0;
}
```

```
void Days(struct Date *p)
```

p



```
struct Date
{
    int year;
    int month;
    int day;
    int yearday;
};
```

```
void Days(struct Date *p)
{
    int monthtable[ ][13]= {
        {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
        {1, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

    int i, leap = 0;
    p -> yearday = p -> day;

    if( ((p -> year %4 == 0) && (p -> year %100 != 0))
                             || (p -> year % 400 == 0) )
        leap = 1;

    for(i = 1; i < p -> month; ++i)
        p -> yearday += monthtable[leap][i];
    printf("所输入的日期是该年的第几天: %d", p -> yearday);
}
```

```
Days(&d1);
```

传址方式-提高效率、“返回”结果

- 例8.2-3 用指针变量操纵结构体。

```
int main( )
{
    struct Date d1;
    scanf("%d%d%d", &d1.year, &d1.month, &d1.day);
    Days(&d1);
    printf("所输入的日期是该年的第几天: %d", d1.yearday);
    return 0;
}
```

yearday
day
month
year

```
void Days(struct Date *p)
```

p

```
struct Date
{
    int year;
    int month;
    int day;
    int yearday;
};
```

```
void Days(struct Date *p)
{
     int monthtable[ ][13]= {
          {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
          {1, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

     int i, leap = 0;
     p -> yearday = p -> day;

     if( (p -> year %4 == 0) && (p -> year %100 != 0))
                              || (p -> year % 400 == 0) )
          leap = 1;

     for(i = 1; i < p -> month; ++i)
          p -> yearday += monthtable[leap][i];
}
```

- 如果不需要通过参数返回数据，则可以用const避免函数的副作用。比如，

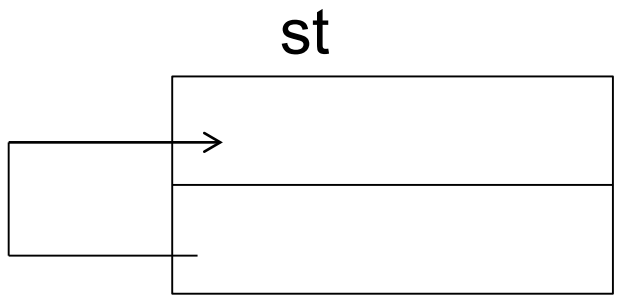
```
void G(const Date *p)
{
  …
  p -> day = 20;   //会出错，因为不能改变p所指向的数据
  …
}
```

- 函数也可以返回一个结构体的地址。

重难点

- 结构类型不可以含有本结构类型成员。
- 结构类型可以含有**本结构类型的指针成员**。比如，

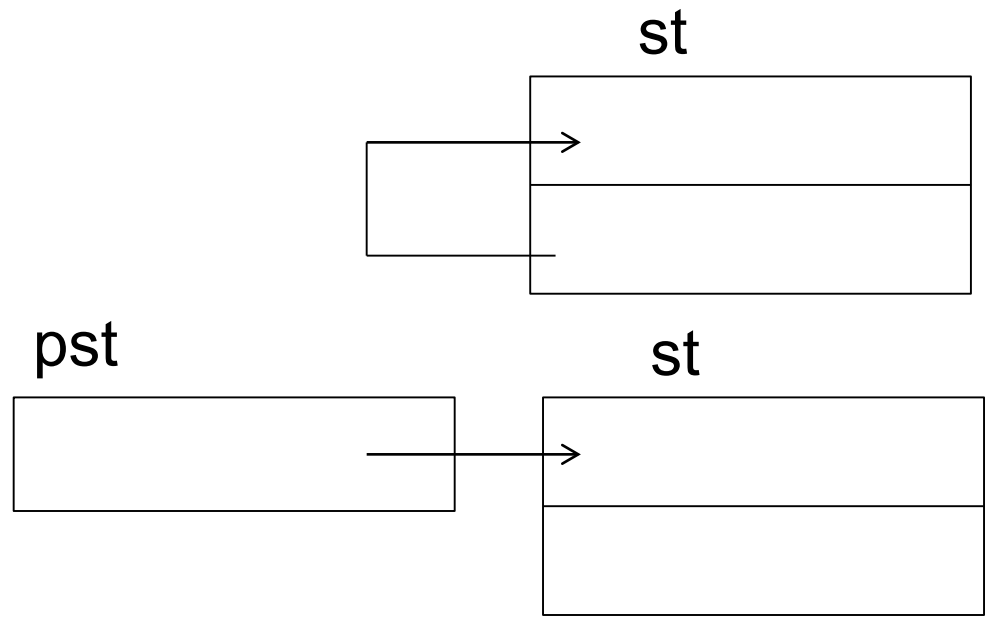
```
struct Stup
{
  int no;
  Stup *p0;
} st;

st.p0 = &st;
```



重难点

- 结构类型不可以含有本结构类型成员。
- 结构类型可以含有**本结构类型的指针成员**。比如，

```
struct Stup
{
  int no;
  Stup *p0;
} st;
```

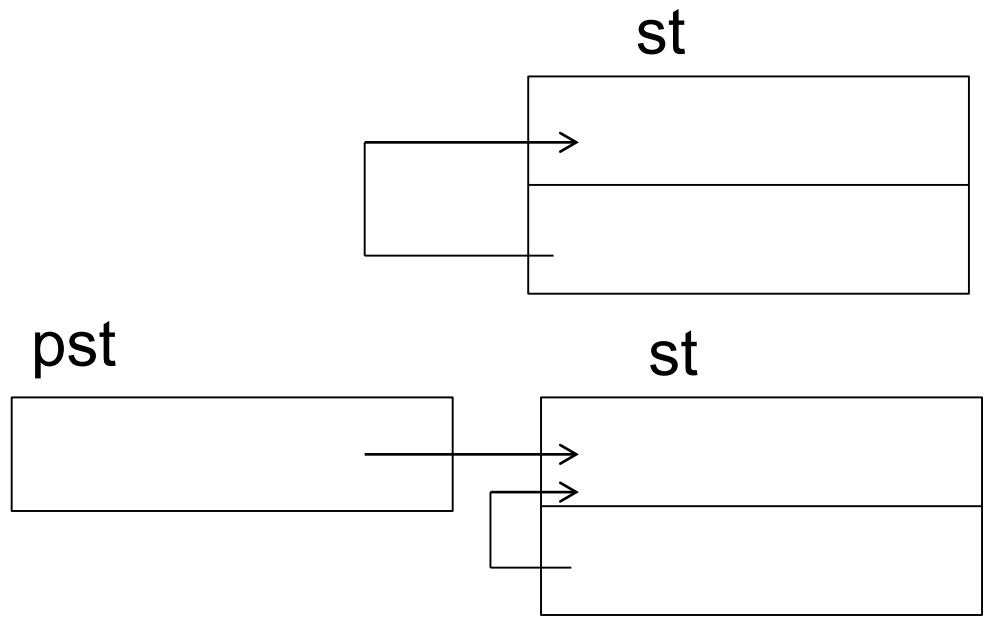



```
st.p0 = &st;
Stup *pst = &st;
```

重难点

- 结构类型不可以含有本结构类型成员。
- 结构类型可以含有本结构类型的指针成员。比如，

```
struct Stup
{
  int no;
  Stup *p0;
} st;
```



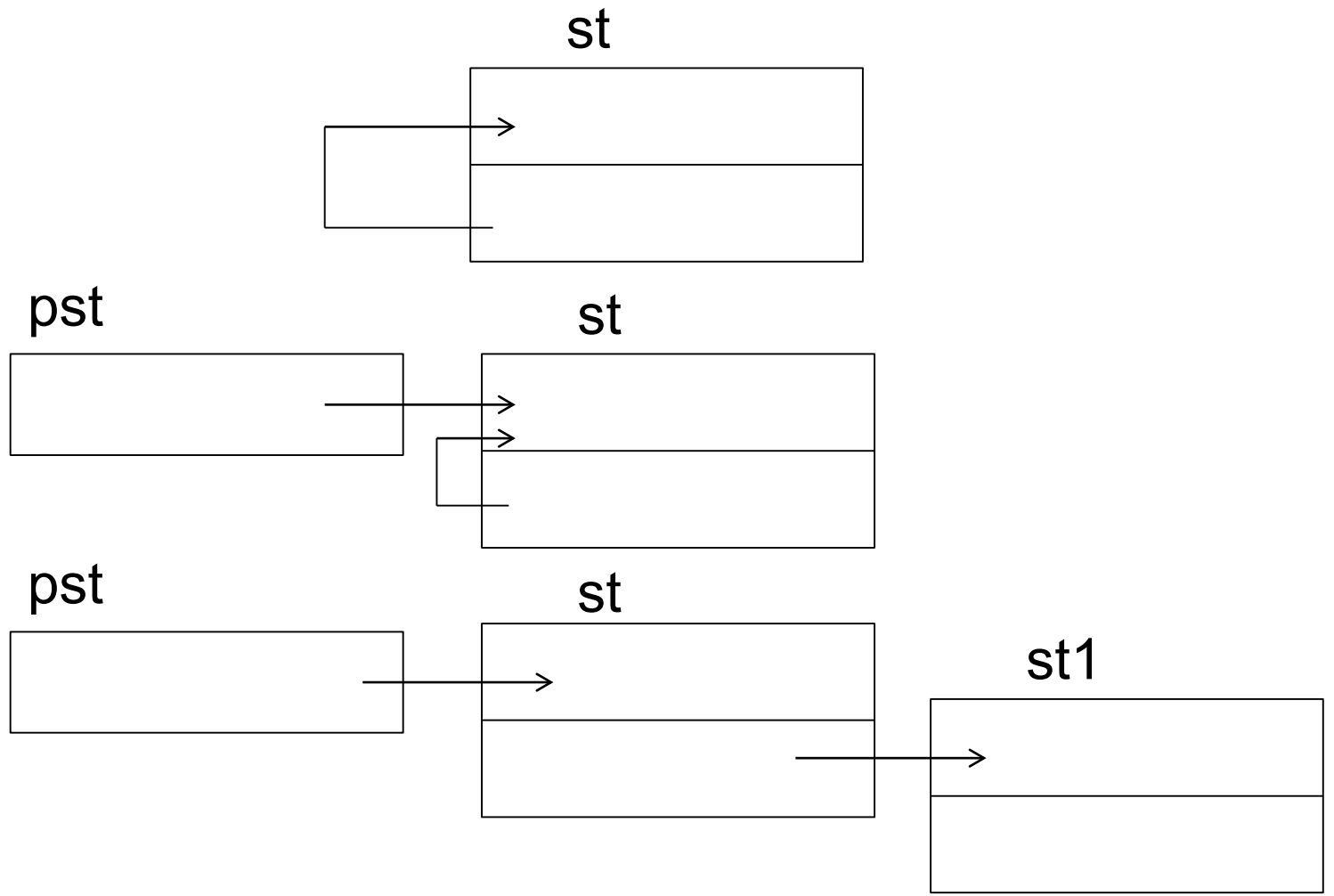

```
st.p0 = &st;
Stup *pst = &st;
pst -> p0 = &st;
```

重难点

- 结构类型不可以含有本结构类型成员。
- 结构类型可以含有本结构类型的指针成员。比如，

```
struct Stup
{
  int no;
  Stup *p0;
} st;
```

```
st.p0 = &st;
Stup *pst = &st;
pst -> p0 = &st;
Stup st1;
pst -> p0 = &st1;
```



小结

- 结构是一种派生数据类型，用来描述多个不同类型的数据群体

- 要求：
  - 掌握结构变量的定义、初始化和操作方法
  - 继续保持良好的编程习惯

Thanks!