计算机程序设计

刘龙军 副教授

2024年10月13日

目录

结构体

一个问题

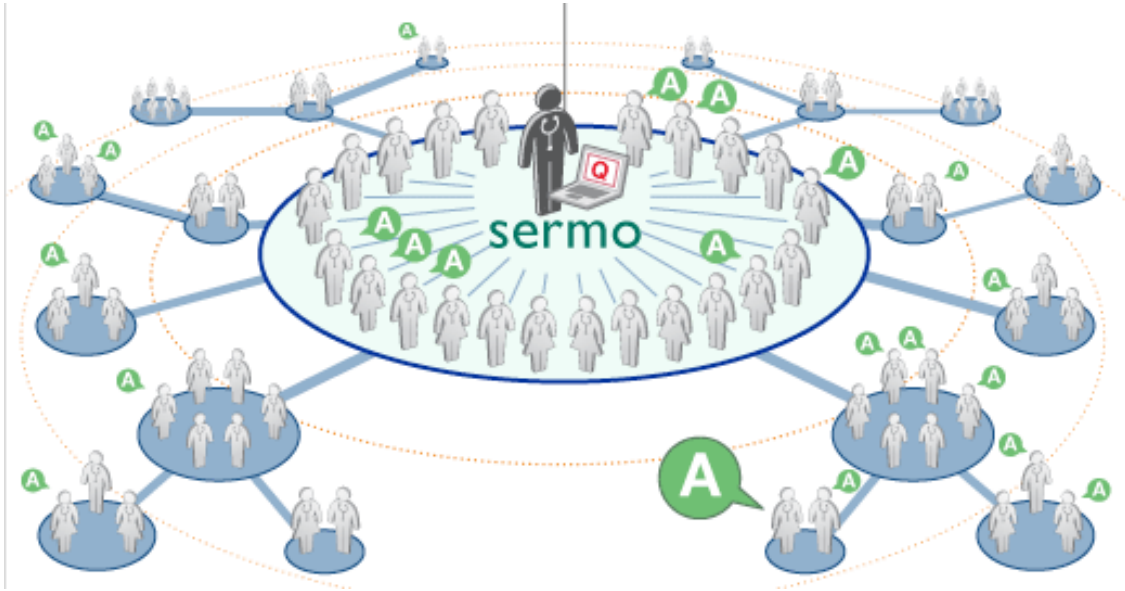
- 在程序里表示一个人（姓名、性别、 年龄、身高、体重……），怎么表示？

```
char name[13];
char sex;
unsigned int age;
。。。。。。
```

- 想表示多个人呢？（数组？）

现实生活中的数据实体

- 在生活中，有着大量由不同性质的数据构成的实体，如，
  - 日期由年、月、日组成；通讯录由姓名、地址、电话、邮政编码等组成；人由姓名、性别、出生年月、通讯录等组成；学生由学号、姓名、性别、课程、成绩等组成；课程由课号、课名、授课教师、授课地点、授课时间等组成; 图形领域的点包含x,y坐标
- 对于类似日期、通信录、人这样的实体，用数组很难描述



解决办法

- 一种称为结构体的**构造型数据类型**。结构体是一组**相关的不同类型**的数据的集合。结构类型为处理复杂的数据提供了便利的手段（**与数组对比**）
- struct person是一个结构类型
- 该结构的成员变量
  - student.name
  - superstars[0].age

```
struct person
{
    char name[12];
    char gender;
    unsigned int age;
 };

struct person student;
struct person superstars[12];
```

结构体的基本知识

- 结构类型定义
- 结构的定义明确了结构的组成形式，定义了一种C/C++语言中**原来没有、而用户实际需要的新的数据类型**。

```
struct 结构类型名
{
    类型 成员名1;
    类型 成员名2;
    ……
    类型 成员名n;
};  //分号不能少
```

例子

- 为了描述日期，可以定义如下结构：

```
struct date
{
   int year;     // 年: 整型作为结构中的成员
   int month;   // 月
   int day;     // 日
};
```

- 为了处理通信录，可以定义如下结构：

```
struct address
{
  char name[30];   // 姓名。字符数组作为结构中的成员
  char street[40]; // 街道名称
  char city[20];   // 城市
  char state[2];   // 省市代码
  unsigned long zip; //邮政编码。无符号长整型作为结构中的成员
};
```

注意事项

- 与其他的数据类型不同，在程序编译的时候结构的定义并不会使系统为该结构分配内存空间，只有在说明结构变量时才分配内存空间。
- 在程序中，结构的定义可以在一个函数的内部，也可以在所有函数的外部
  - 函数内部定义的结构，仅在该函数内部有效
  - 定义在外部的结构，在所有函数中都可以使用。

结构变量的声明

西安交通大学 XI'AN JIAOTONG UNIVERSITY

struct 结构类型 结构变量;

```
struct date
{
   int year;
   int month;
   int day;
};
struct date today;
```

```
struct date
{
   int year;
   int month;
   int day;
} today;
```

- 定义结构便是定义了一种由成员组成的复合类型，而用这种类型说明了一个变量，才会产生具体的实体。
- 声明一个结构变量的作用类似于声明一个int型的变量一样，系统为所声明的结构变量按照结构定义时组成，分配存储数据的实际内存单元。

结构体的嵌套

- 这里定义结构类型为person，man为结构变量。在person结构中成员name是字符串存储姓名，sex为字符型，表示性别，birthday为前面定义的日期date类型的结构。**birthday这个成员就是一个嵌套的结构类型定义。**

```
struct person /* 定义person结构类型 */
{
    char name[30];
    char sex;
    struct date birthday; /* 结构的成员又是结构，即结构的嵌套定义 */
} man;
```

结构体的初始化

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

struct person
{
  char name[30];
  char sex;
  struct date birthday;
};

struct date today = { 2001, 10, 1};
struct person man = { "zhang",'M',{1980,3,28} };
```

访问结构中的成员

- 指明结构成员的符号"."是一种运算符，它的含义是访问结构中的成员这样"today.year"的含义就是访问结构变量today中的名为year的成员。

结构变量.成员变量名

```
today.year=2010; today.month=1; today.day=1;
```

访问结构体中的成员

- 由于结构person中的成员birthday是一个date型的结构，在给birthday赋值时，**不能将birthday作为一个整体**，要用"."运算再**深入一层访问到最基本的成员year、month或day**
- 如果要将"zhang先生"改为"zhong先生"，只要将结构变量man中的数组成员**name下标为2的元素'a'改为'o'即可。**可以使用下列语句：

man.name[2]= 'o';

```
man.sex ='M';
man.birthday.year=1980;
man.birthday.month=3;
man.birthday.day=28;
```

注意事项

- 结构的成员**可以像一般变量一样参与各种操作和运算**
- **而作为代表结构整体的结构变量，** 能够对结构进行整体操作的运算不多，只有进行赋值"="和取地址"&"操作。
  - 例如：

    struct date sunday, today;

    sunday=today;

    进行结构变量整体赋值，就是将结构变量today的值按照各个分量的对应关系赋给结构变量sunday。**这里"="两侧的结构变量类型必须是相同的结构类型**。

例子

例:输入今天的日期，然后输出该日期。

```
//date_struct.cpp
#include <iostream>
using namespace std;

int main()
{
    struct date
    {
        int year;
        int month;
        int day;
    };
    struct date today; /* 说明结构变量today */

    cin>>today.year>>today.month>>today.day;
    cout<<"Year: "<<today.year
        <<" Month: "<<today.month
        <<" Day: "<<today.day
        <<endl;
   return 0;
}
```

注意

- 结构体类型的变量在内存依照其成员的顺序顺序排列，所占内存空间的大小是其全体成员所占空间的总和
- 在编译时，仅对变量分配空间，不对类型分配空间
- 对结构体中各个成员可以单独引用、赋值，其作用与变量等同

  格式：变量名．成员名　student1．num
- 不能对结构体变量整体赋值或输出，只能分别对各个成员引用

```
cin>>student1;
cin>>student1.num;     student1.num=100;
```

注意

- 可以将一个结构体变量整体赋给另外一个相同类型的结构体变量。

  student2=student1;

- 嵌套的结构体变量必须逐层引用

  student1.birthday.day=25;

- 结构体变量中的成员可以同一般变量一样进行运算

  student1.birthday.day++;　　student1.score+=60;

结构体数组

结构体数组

- 结构体与数组的关系有两重：
  - 在结构中使用数组类型作为**结构的一个成员**
  - 用**结构类型作为数组元素**的类型构成数组

```
struct person
{
    char name[30];
    char sex;
    struct date birthday;
} man;

struct person student[100];
```

结构体数组中成员变量 访问

- 同一般的数组一样，结构数组中每个元素的起始下标从0开始，数组名称表示该结构数组的存储首地址。
- 结构数组存放在一连续的内存区域中，它所占内存数目为结构类型的大小乘以数组元素的个数。

结构数组名[下标].成员名

例子

- 例如，我们要将数组man中的3号元素赋值为:"Peter",'M',1980,3,1，就可以使用下列语句：

```
struct person man[100];

strcpy(man[3].name, "Peter");
man[3].sex='M';
man[3].birthday.year=1980;
man[3].birthday.month=3;
man[3].birthday.day=1;
```

结构体与指针

结构体指针

指针结构与指针的关系亦有两重

- 在定义结构时，**将指针作为结构中的一个成员**
- **定义指向结构体的指针**（称为结构体指针）

```
struct somebody
{
    char name[30];
    int score;
    int *ptr;
};

struct person *man;
```

结构体指针

- 结构体指针是指向一种结构类型的指针变量，它是结构体在内存中的首地址，结构指针具有一般指针的特性，如在一定条件下两个指针可以进行比较，也可以与整数进行加减

  但在指针操作时应注意：进行地址运算时的放大因子由所指向的结构的实际大小决定

```
struct 结构体类型 *结构体变量;
```

```
struct date *today;
struct address *home;
```

结构体指针

- 怎样通过ppt访问pt的成员？

  (1) (*ppt).x = 0;

  (2) ppt->x = 0;

- 通过结构指针访问成员可以采用运算符"->"进行操作

```
struct point
{
  int x;
  int y;
};

struct point pt;
struct point* ppt = &pt;
```

例子

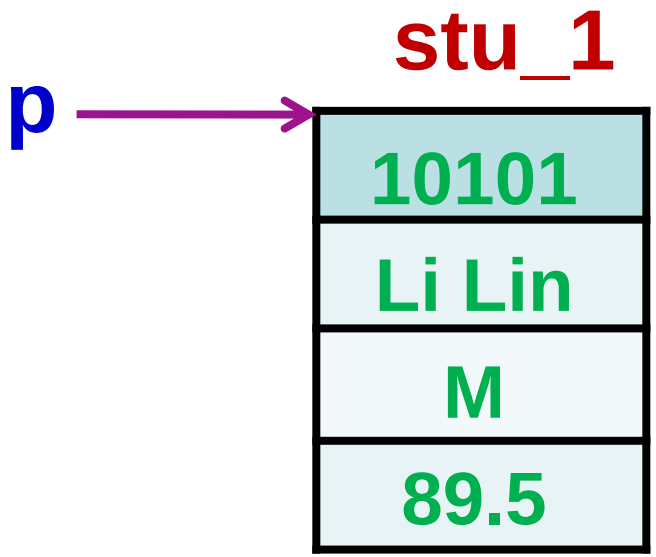
- 下面表达式哪些合法？
  - （1）rt.pt1.x
  - （2）rp.pt1.x
  - （3）(*rp).pt1.x
  - （4）rp->pt1.x
  - （5）rt->pt1.x

```
struct point
{
  int x;
  int y;
};
struct rect
{
  struct point pt1;
  struct point pt2;
};

struct rect rt;
struct rect *rp = &rt;
```

例子

```
struct Student stu_1;
struct Student * p;
p=&stu_1;
stu_1.num=10101;
strcpy(stu_1.name, “Li Lin” );
stu_1.sex='M ‘;  stu_1.score=89.5;
```



指向结构体数组的指针

- 有3个学生的信息，放在结构体数组中，要求输出全部学生的信息。

例子

```
#include <stdio.h>
struct Student
{ int num;    char name[20];
   char sex;   int age;
};
struct Student stu[3]={
            {10101,"Li Lin",'M',18},
            {10102,"Zhang Fun",'M',19},
            {10104,"Wang Min",'F',20} };
```

例子

```
int main()
{ struct Student *p;
   printf(" No.  Name        sex  age\n");
   for(p=stu;p<stu+3;p++)
      printf("%5d %-20s %2c %4d\n",
                  p->num, p->name,
                     p->sex, p->age);
   return 0;
}
```

```
No.  Name                    sex age
```

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

例子

```
 No.  Name                 sex age
10101 Li Lin                M   18
```

```
int main()
{ struct Student *p;
   printf(" No.  Name        sex  age\n");
   for(p=stu;p<stu+3;p++)
      printf("%5d %-20s %2c %4d\n",
                  p->num, p->name,
                     p->sex, p->age);
   return 0;
}
```

p →

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

例子

```
 No.  Name                 sex age
10101 Li Lin                M   18
10102 Zhang Fun             M   19
10104 Wang Min              F   20
```

```
int main()
{ struct Student *p
   printf(" No.  Name          sex  age\n");
   for(p=stu;p<stu+3;p++)
      printf("%5d %-20s %2c %4d\n",
                  p->num, p->name,
                      p->sex, p->age);
   return 0;
}
```

p →

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

用结构体变量和结构体变量的指针作函数参数

- 将一个结构体变量的值传递给另一个函数，有3个方法。回顾一下。

- 用结构体变量的成员作参数
  - 例如，用stu[1].num或stu[2].name作函数实参，将实参值传给形参
  - 用法和用普通变量作实参是一样的，属于“值传递”方式
  - 应当注意实参与形参的类型保持一致

（2）

- 用结构体变量作实参。
  - 用结构体变量作实参时，将结构体变量所占的**内存单元的内容全部按顺序传递给形参**，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。**这种传递方式在空间和时间上开销较大**
  - 在被调用函数期间改变形参（也是结构体变量）的值，不能返回主调函数
  - 一般较少用这种方法

（3）

- 用指向结构体变量（或数组元素）的**指针作实参**，将结构体变量（或数组元素）的地址传给形参。

例子

- 例　有n个结构体变量，内含学生学号、姓名和3门课程的成绩。

　　要求输出平均成绩最高的学生的信息(包括学号、姓名、3门课程成绩和平均成绩)。

例子

- 解题思路：将n个学生的数据表示为结构体数组。按照功能函数化的思想，分别用3个函数来实现不同的功能：
  - 用input函数输入数据和求各学生平均成绩
  - 用max函数找平均成绩最高的学生
  - 用print函数输出成绩最高学生的信息
  - 在主函数中先后调用这3个函数，用指向结构体变量的指针作实参。最后得到结果。
  - 本程序假设n=3

例子

```
#include <stdio.h>
#define N 3
struct Student
{  int num;
    char name[20];
    float score[3];
    float aver;
};
```

4个成员

输入前3个成员值

计算最后成员值

例子

```
void input(struct Student stu[]);
struct Student max(struct Student stu[]);
void print(struct Student stu);

int main()
{
   struct Student stu[N],     *p=stu;
   input(p);
   print(max(p));
   return 0;
}
```

例子

```
void input(struct Student stu[])
{ int i;
   printf("请输入各学生的信息：
                学号、姓名、三门课成绩:\n");
   for(i=0;i<N;i
   {scanf("%d %s %
              &stu[i].num,stu[i].name,
          &stu[i].score[1],
           &stu[i].score[2]);
     stu[i].aver=(stu[i].score[0]+
        stu[i].score[1]+stu[i].score[2])/3.0;
   }
}
```

i=1

输入第1个成员

输入第2个成员值

输入第3个成员值

计算第4个成员值

stu

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
						stu[2]

例子

```
void input(struct Student stu[])
{ int i;
   printf("请输入各学生的信息：
                学号、姓名、三门课成绩:\n");
   for(i=0;i<N;i
   {scanf("%d %s %
              &stu[i].num,stu[i].name,
           &stu[i].score[1],
            &stu[i].score[2]);
     stu[i].aver=(stu[i].score[0]+
        stu[i].score[1]+stu[i].score[2])/3.0;
   }
}
```

i=2

输入第1个成员

输入第2个成员值

输入第3个成员值

计算第4个成员值

stu

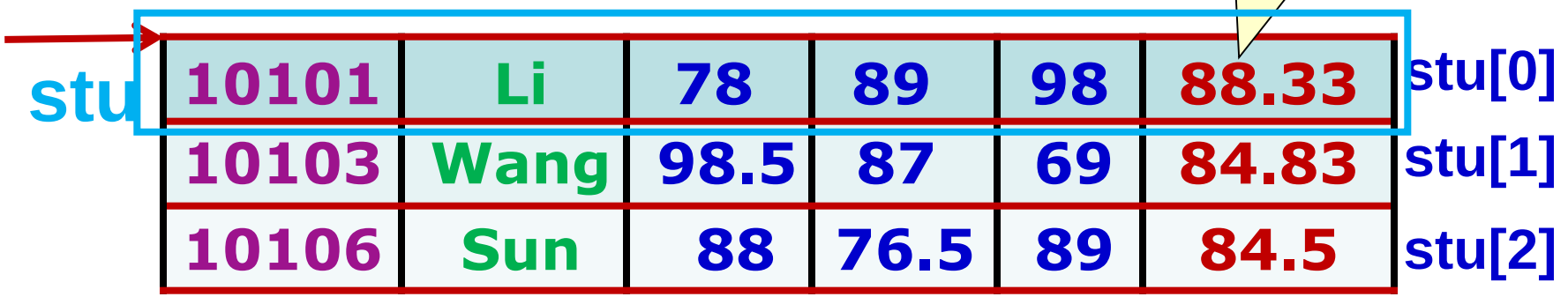
10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
10106	Sun	88	76.5	89	84.5	stu[2]

例子

```
struct Student max(struct Student stu[])
{int i,m=0;
  for(i=0;i<N;i++)
     if (stu[i].aver>stu[m].aver) m=i;
  return stu[m];
}
```

返回

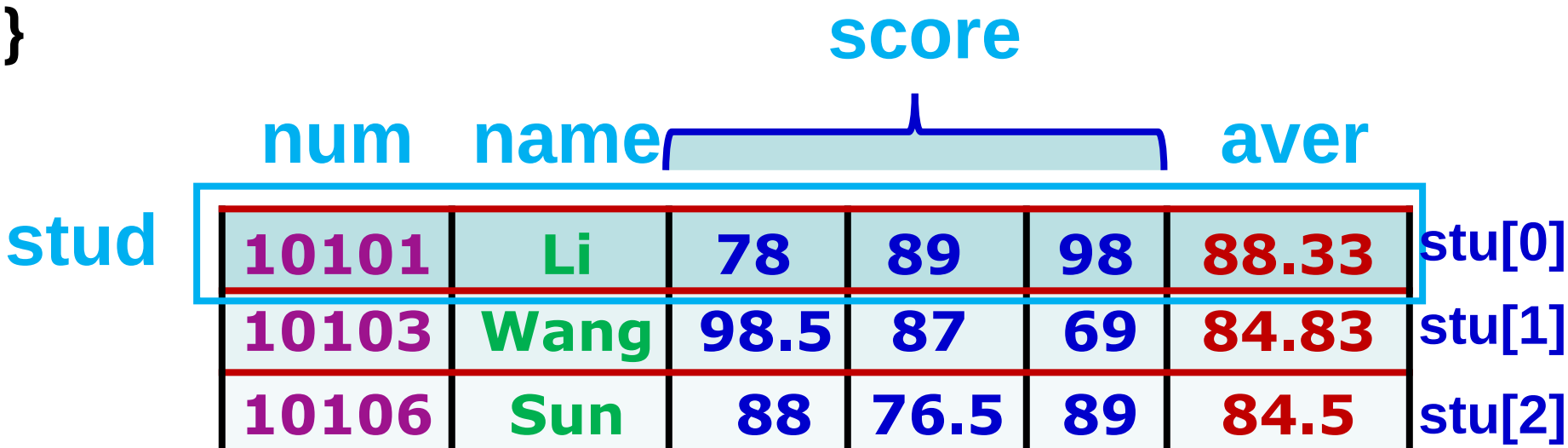
最大



例子

```
void print(struct
{ printf("\n成绩最
   printf("学号:%d
        三门课成绩:%5.1f,%5.1f,%5.1f\n
          平均成绩:%6.2f\n”, stud.num,
           stud.name,stud.score[0],
     stud.score[1],stud.score[2],stud.aver);
}
```

```
成绩最高的学生是:
学号:10101
姓名:Li
三门课成绩: 78.0, 89.0, 98.0
平均成绩: 88.33
```



例子

- 以上3个函数的调用，情况各不相同：
  - 调用input函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数无返回值。
  - 调用max函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数的返回值是结构体类型数据。
  - 调用print函数时，实参是结构体变量，形参是结构体变量，传递的是结构体变量中各成员的值，函数无返回值。

* 链表的表示（将来数据结构课程的重点，动态！）


```
struct something
{
  char name[10];
  struct something* pnext_obj;
};
```

定义自己的类型名

- 标准类型有：
  - int、char、float、double、long
  - 以及结构体(struct)、共用体(union)、枚举类型(enum)等
- 此外还可以定义新的类型来代替已有的类型名，如：
  - typedef int INTEGER; 指定INTEGER代表int类型
  - typedef float REAL; 指定REAL代表float类型

例子

```
struct person
{
    char name[12];
    char sex;
};

struct person student;
/* It works */
person student;
/* Can this work? */
```

```
typedef struct
{
  char name[12];
  char sex;
} person;

person student;
/* It works! */
```

枚举类型

枚举类型

- 如果一个**变量只有几种可能的值**，则可以定义为**枚举类型**
- 所谓“枚举”就是指把可能的值一一列举出来，**变量的值只限于列举出来的值的范围内**

使用枚举类型例

- 声明枚举类型用enum开头（enumerate）
- 例如：
- enum Weekday{sun,mon,tue, wed,thu,fri,sat};
- 声明了一个枚举类型enum Weekday
- 然后可以用此类型来定义变量
- enum Weekday workday,weekend ;

枚举元素

枚举变量

例子

- workday=mon;　　正确
- weekend=sun;　　正确
- weekday=monday;　　不正确

注意

- 编译器对枚举类型的**枚举元素按常量处理，故称枚举常量**。不要因为它们是**标识符(有名字)**而把它们看作变量，**不能对它们赋值**。例如:
- sun=0; mon=1; **错误**

注意

- 每一个枚举元素都代表一个整数，编译器按定义时的顺序默认它们的值为0,1,2,3,4,5…
- 在上面定义中，sun的值为0，mon的值为1,…sat的值为6
- 如果有赋值语句: workday=mon;
- 相当于workday=1;

说明

- 每一个枚举元素都代表一个整数，编译器按定义时的顺序默认它们的值为0,1,2,3,4,5...
- 也可以人为地指定枚举元素的数值，例如:
  - enum Weekday{sun=7,mon=1,tue,wed,thu,fri,sat}workday,week_end;
  - 指定枚举常量sun的值为7，mon为1，以后顺序加1，sat为6。

注意

- 枚举元素可以用来作判断比较。例如：
  - if(workday==mon)…
  - if(workday>sun)…
  - 枚举元素的**比较规则是按其在初始化时指定的整数来进行比较的**。
  - 如果定义时未人为指定，则按上面的默认规则处理，即第一个枚举元素的值为0，故mon>sun，sat>fri

例子

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    enum color { red=1, green, blue };
    enum  color favorite_color;

    /* ask user to choose color */
   printf("请输入你喜欢的颜色: (1. red, 2. green, 3. blue): ");
   scanf("%d", &favorite_color);
```

例子

```
switch (favorite_color)
    {
    case red:
        printf("你喜欢的颜色是红色");        break;
    case green:
        printf("你喜欢的颜色是绿色")         break;
    case blue:
        printf("你喜欢的颜色是蓝色");        break;
    default:
        printf("你没有选择你喜欢的颜色");
    }
    return 0;
}
```

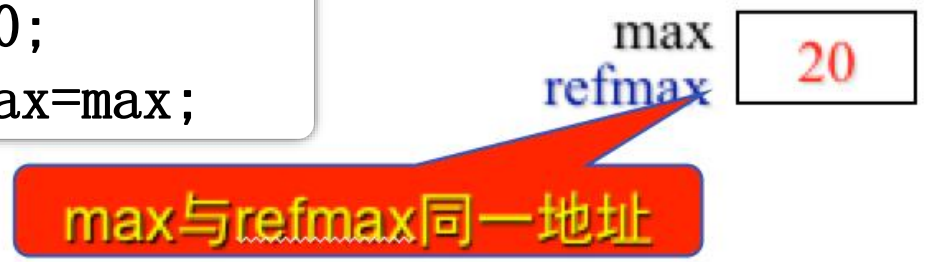
引用

引用定义

- 引用是变量的别名（小名）
- 对引用操作等同于对原始变量的操作
- 其中原变量名必须是一个已定义过的变量
- max与refmax在内存中占用同一地址，即同一地址两个名字

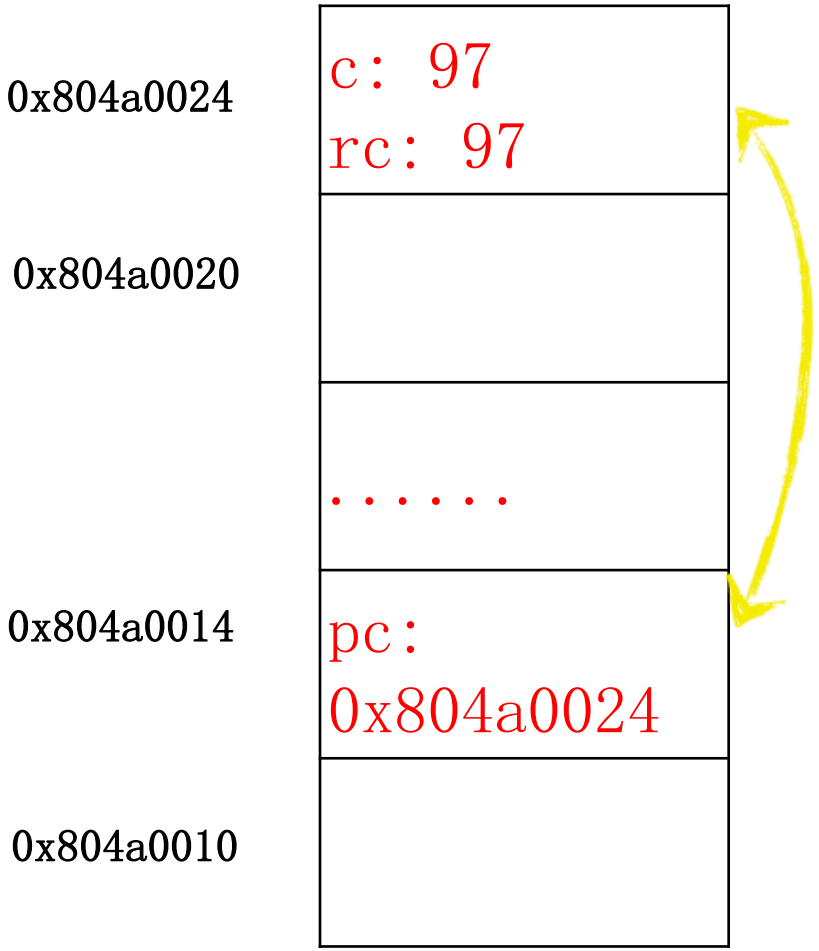
类型 &标识符

```
int max=20;
int &refmax=max;
```



指针/引用所指变量

```
char c ='a';
char *pc = &c;
char &rc = c;
```



引用和指针的区别

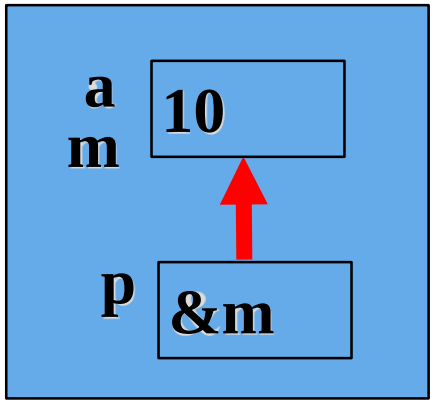
- 引用在定义的时候必须要初始化
- 引用一旦被初始化后不得再作为其它变量的别名引用

```
int rats = 101;
int & rodents = rats;    // rodents a reference
int * prats = &rats;     // prats a pointer

int rat;
int & rodent;
rodent = rat;    // No, you can't do this.
```

引用和指针的关系

```
int a;
int &m=a;
int *p;
p=&m;
*p=10;
```



当&a的前面有类型符时(如int &a)，它必然是对引用的声明；如果前面无类型符(如cout<<&a;)，则是取变量的地址

对引用执行取地址操作，结果只能是所引用的变量地址

不合法的引用

- 企图建立引用的数组　　int & a[9];
- 企图建立指向引用的指针　　int & *p;
- 企图建立引用的引用　　int & &px;
- 企图建立void应用　　void & a=x;
- 用类型或NULL来初始化　　int & a=int;　int & a =NULL;

引用作为函数参数

- 引用的主要作用是作为函数的参数

```
a = 123; b = 456
a = 456; b = 123
```

```
//swap_ref.cpp
……
int main()
{
    int a=123;
    int b=456;

    int &refa=a;
    int &refb=b;

    cout<<"a = "<<a<<"; b = "<<b<<endl;
    Swap(refa,refb);     //exchange a,b
    cout<<"a = "<<a<<"; b = "<<b<<endl;

    return 0;
}

void Swap(int &x, int &y)
{
    int tmp;
    tmp = x;
    x = y;
    y= tmp;
}
```

小结数据类型

- **类型本不存在** ： 内存里存储的内容，你认为它是什么，它就是什么
- 高级语言设计了基本数据类型：整型、浮点型、字符型等。不同的语言也会定义不同的基本类型，但是**基本数据类型并不能方便地解决所有问题**
- 复合数据类型是基本数据类型迭代派生而来，典型的代表就是**“结构体”(structure)**，数组、指针也可算作此类
- 抽象数据类型(ADT)在复合数据类型的基础上**增加了对数据的操作**
- **抽象数据类型进而进化为“类(Class)”这是一个跨时代的进步**

本章结束！