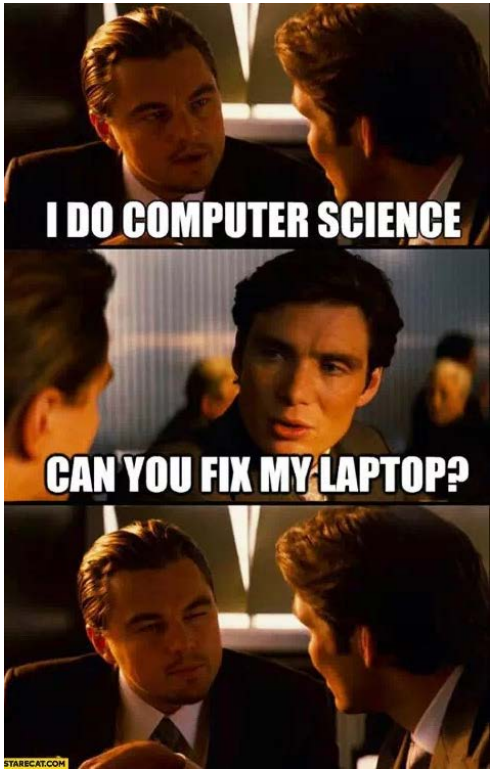
第9章 建立自己的数据类型（5）

S.IST@XMU

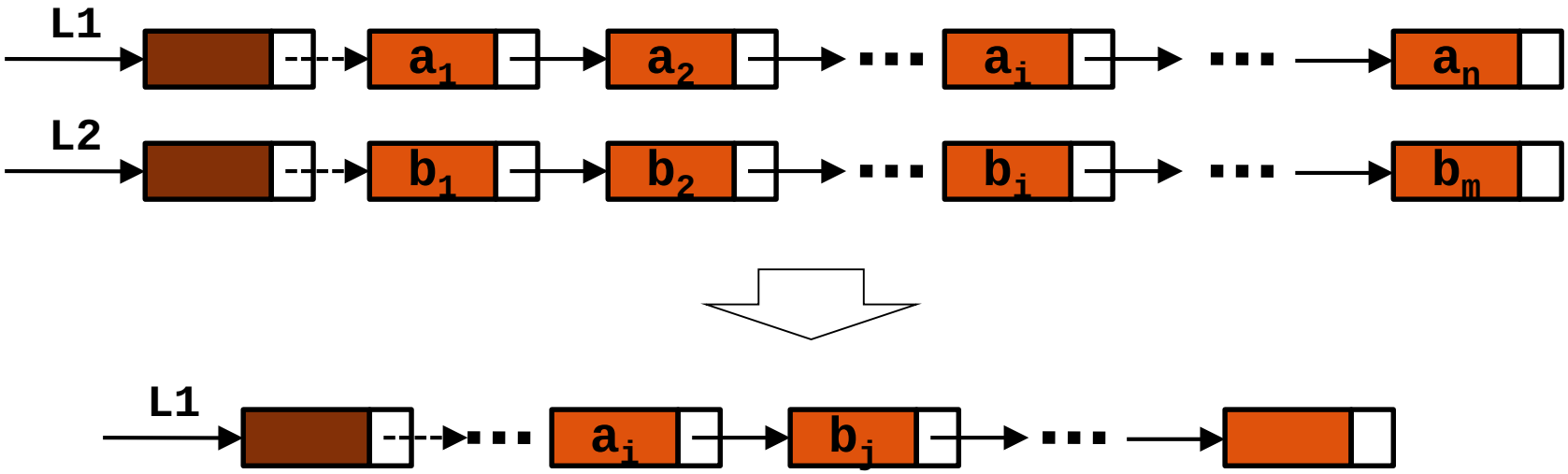
复习回顾

- 上次课的内容：
  - ◆建立静态链表
  - ◆建立动态链表
  - ◆链表的操作
    - ●插入
      - ✪ 带表头节点
      - ✪ 不带表头节点
    - ●删除
  - ◆天冷了，快放寒假了，有个事情要有心理准备……



带头结点的有序链表的合并

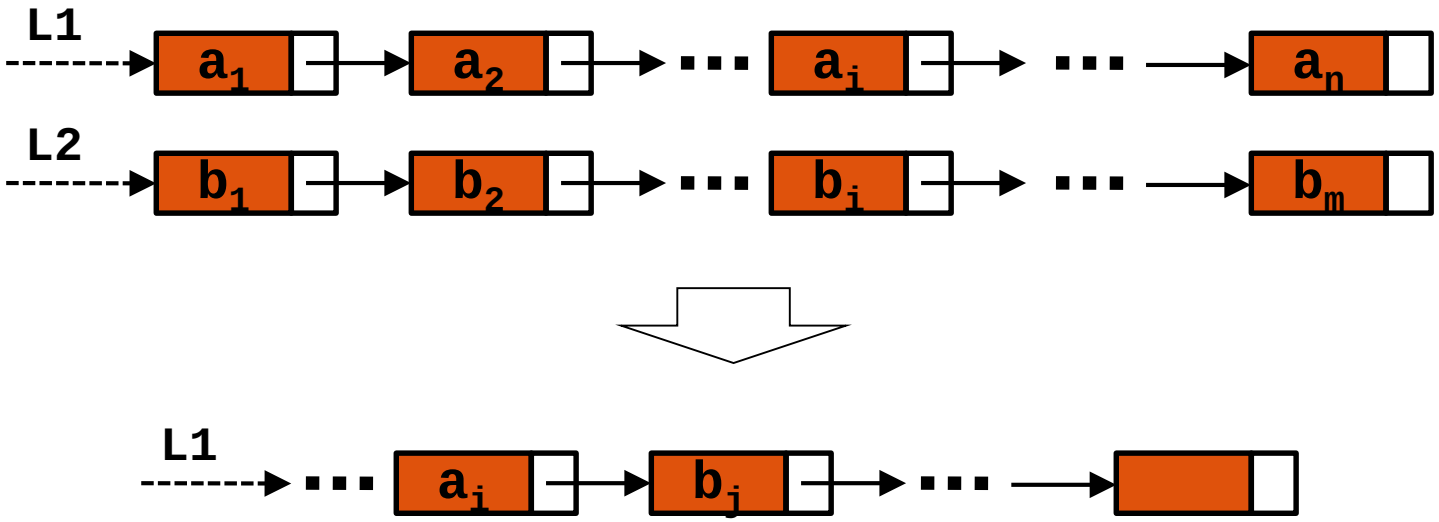
- 如图所示，链表L1和L2的结点数据顺序递增，写一个函数把L2合并到L1中，且保持结点数据的顺序递增。



```
1.  struct node * mergenodes(struct node *L1, struct node *L2)
2.  {
3.      struct node *p1=L1, *q1=L1->next, *p2=L2->next, *q2=L2->next;
4.      while (p2 != NULL)
5.      {
6.              q2 = p2->next; //用q2记下p2在L2中的下一个结点
7.              while (q1!=NULL && q1->data < p2->data)
8.              {
9.                      p1 = q1; q1 = q1->next;
10.             }
11.              //如果已经到达L1表尾，可直接把L2其余部分链接到L1尾部，不需循环
12.             if (q1 == NULL)
13.                     break;
14.             p2->next = q1;
15.             p1->next = p2;
16.             p1 = p2;
17.             p2 = q2;          //p2指向L2中的下一个结点
18.     }
19.     if (q1 == NULL) //如果已经到达L1表尾，可直接把L2其余部分链接到L1尾部
20.     {
21.             p1->next = p2;
22.     }
23.     return L1;
24. }
```

不带头结点的有序链表的合并

- 如图所示，链表L1和L2的结点数据顺序递增，写一个函数把L2合并到L1中，且保持结点数据的顺序递增。



```
1.  struct node * mergenodes(struct node *L1, struct node *L2)
2.  {
3.      struct node *p1=L1, *q1=L1, *p2=L2, *q2=L2;
4.      while (p2 != NULL)
5.      {
6.              q2 = p2->next; //用q2记下p2在L2中的下一个结点
7.              while (q1!=NULL && q1->data < p2->data)
8.              {
9.                      p1 = q1; q1 = q1->next;
10.             }
11.              //如果已经到达L1表尾，可直接把L2其余部分链接到L1尾部，不需循环
12.             if (q1 == NULL)
13.                     break;
14.             if (q1 == L1) //若插入位置在L1的表头，需修改L1
15.             {
16.                     p2->next = q1;  p1 = q1 = L1 = p2;
17.             }
18.             else //若插入位置在L1的其他位置
19.             {
20.                     p2->next = q1; p1->next = p2;
21.                     p1 = p2;
22.             }
23.             p2 = q2; //p2指向L2中的下一个结点
24.     }// 未完待续
```

```
25.     //紧接上页
26.     if (q1 == NULL) //如果已经到达L1表尾，可直接把L2其余部分链接到L1尾部
27.     {
28.             if (p1 == NULL)
29.             {
30.                     L1 = p2;
31.             }
32.             else
33.             {
34.                     p1->next = p2;
35.             }
36.     }
37.
38.     return L1;
39. }
```

查找不带头结点的链表中的结点

```
1.  struct node * findnode(struct node *head, int key)
2.  {
3.      // 临时指针p指向head为头指针的链表的首结点
4.      struct node * p=head;
5.      // 当p尚未指向链表末尾
6.      while (NULL != p)
7.      {
8.          //检查当前p指向的结点数据data是否等于要查找的key
9.          if (p->data == key)
10.         {
11.             //若找到，跳出while循环
12.             break;
13.         }
14.         //若当前p指向的结点数据不等于key，p转而指向下一个结点
15.         p = p->next;
16.     }
17.     // 此时若找到，则p指向data为key的结点；若没找到，p的值NULL
18.     return p;
19. }
```

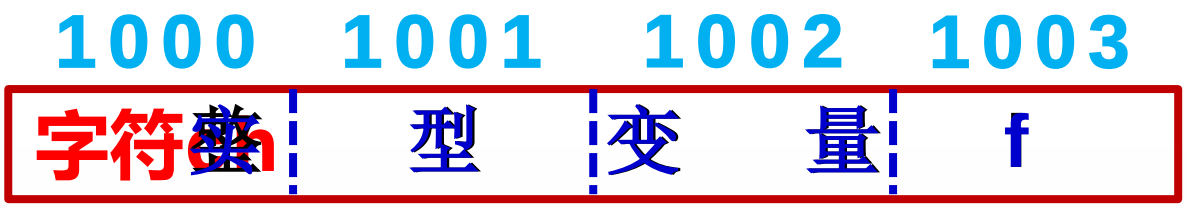
对链表中的结点进行冒泡排序

```
1.  void bubblesort(struct node *head)
2.  {
3.      struct node * tail=NULL, *p, *q;
4.      if (NULL == head) return NULL;
5.      do
6.      {
7.              p = head;
8.              q = p->next;
9.              while (tail != q)
10.             {
11.                     if (p->data > q->data)
12.                     {
13.                             swapdata (p, q);
14.                     }
15.                     p = q;
16.                     q = p->next;
17.             }
18.             tail = p;
19.     } while (tail != head);
20. }
```

```
1.  void swapdata(struct node *p,
2.                    struct node *q)
3.  {
4.      int tmp = p->data;
5.      p->data = q->data;
6.      q->data = tmp;
7.  }
```

什么是共用体类型

- 现实生活中某些事物往往可以用多种方式表述
  - ◆称谓：可用姓名，字，号，官职
  - ◆成绩：可用数字，也可用“优” “良” “中” “差”
- 有时想用同一段内存单元存放不同类型的变量
  - ◆可以比数组，或比结构体都节省空间
- 使几个不同的变量共享同一段内存的结构，称为“共用体”类型的结构。



共用体类型定义的一般形式

- 定义共用体类型变量的一般形式为：

union 共用体名 { 成员表列 } 变量表列；

例如：

```
union Data
{
    int i;
    char ch;
    float f;
}a,b,c;
```

```
union
{
    int i;
    char ch;
    float f;
} a,b,c;
```

```
union Data
{
    int i;
    char ch;
    float f;
};
union Data a,b,c;
```

貌合神离的结构体与共用体

- **相似1：先定义后引用。共用体变量可以使用”.”运算符引用其成员，但不能直接引用共用体变量**
  - ◆以下是正确的
    - scanf ("%s",xiaoming.degree);
    - xiaoming.score=90;
    - xiaoming = yao;
  - ◆以下是错误的
    - printf("%d,%s",xiaoming);

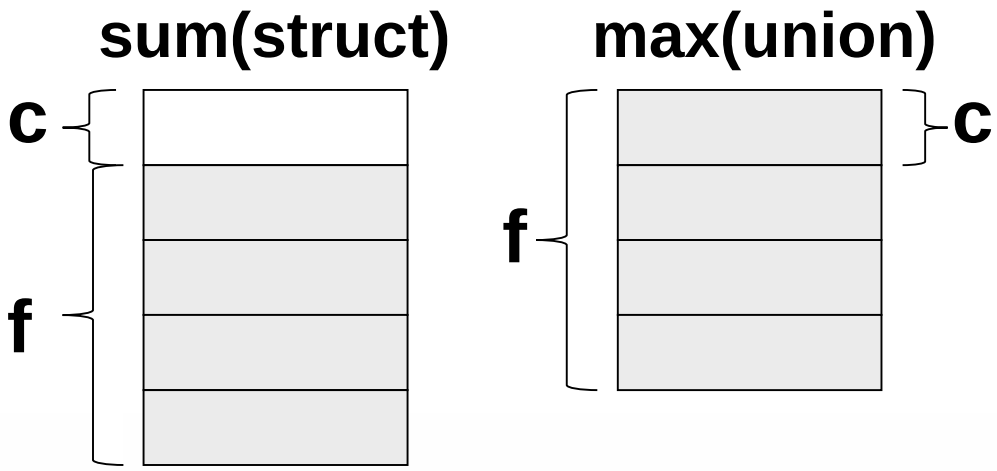
```
union mark
{
    int score;
    char degree[4];
} xiaoming, yao;
```

- **相似2：系统给共用体变量分配空间，不为共用体类型分配空间**

貌合神离的结构体与共用体

- **不同1：可以简单地认为，结构体变量所占内存长度是各成员占的内存长度之和，每个成员分别占有其自己的内存单元。而共用体变量所占的内存长度等于最长的成员的长度。**

```
struct
{
   char c;
   float f;
} sum;
```




```
union
{
   char c;
   float f;
} max;
```

貌合神离的结构体与共用体

- 不同2：初始化方式截然不同。

```
union mark
{
   int score;
    char degree[4];
} xiaoming={90, "优"};    //出错
```

```
union mark
{
   int score;
    char degree[4];
} xiaoming={90};          //正确初始化
```

共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

  (1)同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一个成员，而不是同时存放几个。

  - 可以想象一下，一间教室不会同时有多位老师在讲课……

共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

(2)可以对共用体变量初始化，但初始化表中只能有一个常量，只需初始化第一个成员。

```
union  //正确
{
   char c;
   float f;
} max={'a'};
```

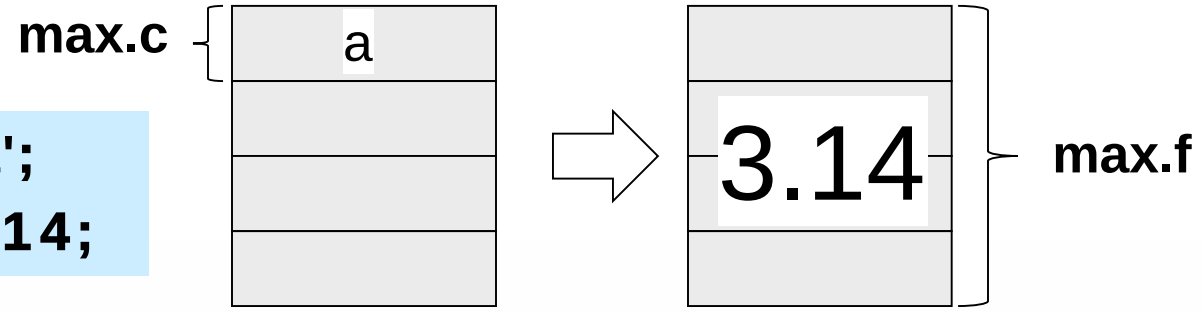
```
union  //编译通过，结果错误
{
   char c;
   float f;
} max={3.14};
```

共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

(3)共用体变量中起作用的成员是最后一次被赋值的成员，在对共用体变量中的一个成员赋值后，原有变量存储单元中的值就取代。

```
max.c = 'a';
max.f = 3.14;
```



共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

(4) 共用体变量的地址和它的各成员的地址都是同一地址。

◆&max, &max.c, &max.f 是**相等的**

(5) 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值。

◆max=3.14; 或 int x=max; 都**错误**

共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

（6）以前的C规定不能把共用体变量作为函数参数，但可以使用指向共用体变量的指针作函数参数。C99允许用共用体变量作为函数参数。

```
union mark *pmark;        pmark=&xiaoming;
int s=pmark->score;       //相当于s=xiaoming.score
void foo (union mark * pmark);   //ok
void foo (union mark m); //仅在C99的标准允许
```

共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

（7）共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

```
union U {int a; struct S s; }; //共用体的成员可以是结构体
struct S {int a; union U u; }; //结构体的成员可以是共用体
```

共用体应用例子

例：有若干个人员的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。要求用同一个表格来处理。

➢解题思路：

◆学生和教师的数据项目多数是相同的，但有一项不同。现要求把它们放在同一表格中

num	name	sex	job	class(班) / position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

➢解题思路：

◆如果job项为s，则第5项为class。即Li是501班的。如果job项是t，则第5项为position。Wang是prof（教授）。

num	name	sex	job	class(班) position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

➢解题思路：

◆对第5项可以用共用体来处理（将class和position放在同一段存储单元中）

num	name	sex	job	class(班) position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

第一种定义方式

```
#include <stdio.h>
struct
{
    int num;
    char name[10];
    char sex;
    char job;
    union
    {
        int clas;
        char position[10];
    } category;
} person[2];
```

共用体变量

外部的结构体数组

第二种定义方式

```
#include <stdio.h>
union Categ
{
    int clas;
    char position[10];
};
struct
{
    int num;
    char name[10];
    char sex;
    char job;
    union  Categ  category
} person[2];
```

声明共用体类型

定义共用体类型变量

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i;
5.     for (i=0;i<2;i++)
6.     {
7.         scanf("%d %s %c %c",&person[i].num,
8.                             &person[i].name,
9.                             &person[i].sex,
10.                            &person[i].job);
11.        if (person[i].job == 's')
12.            scanf("%d",&person[i].category.clas);
13.        else if (person[i].job == 't')
14.            scanf("%s",person[i].category.position);
15.        else
16.            printf("Input error!");
17.    }
18.    printf("\n");     //未完待续……
```

```
    // 紧接上页
    for (i=0;i<2;i++)
    {
        if (person[i].job == 's')
            printf("%-6d%-10s%-4c%-4c%-10d\n",
                    person[i].num, person[i].name,
                    person[i].sex, person[i].job,
                    person[i].category.clas);
        else
            printf("%-6d%-10s%-4c%-4c%-10s\n",
                    person[i].num, person[i].name,
                    person[i].sex, person[i].job,
                    person[i].category.position);
    }
    return 0;
}
```

“表里不一”的枚举类型

- 形式上像构造类型，实际上是基本类型
- 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”（enumerate）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

枚举类型的作用

- 更自然地表示**非数值计算**中的数据
  - **存在问题：**用数值表示诸如性别、月份、星期、颜色这样的数据，既不直观也不易阅读
  - **解决方法：**用自然语言中含义与其相对应的单词来表示某一状态，程序就容易阅读和理解
  - **注意：**对编译器来说，这些单词不是字符串，而是整型这样的基本类型

如何声明枚举类型

- 声明枚举类型用enum开头。
- 例如：

枚举元素

```
enum Weekday{sun,mon,tue,
                    wed,thu,fri,sat};
```

◆声明了一个枚举类型enum Weekday

```
enum Color {red,green,blue};
```

◆声明了一个枚举类型Color

如何定义枚举类型变量

- 与结构体和共同体类似，也分成三种形式
  - ◆先定义枚举类型，再定义枚举变量

    enum 枚举名{枚举值表};

    enum 枚举名 变量名列表;

```
enum Color{red,green,blue};
enum Color cl;
```

枚举变量

  - ◆定义枚举类型的同时定义枚举变量

    enum 枚举名{枚举值表}变量名列表;

```
enum Color{red,green,blue} cl;
```

  - ◆不使用枚举名，直接定义枚举变量

    enum {枚举值表} 变量名列表;

```
enum {red,green,blue} cl;
```

枚举类型变量的赋值

- 枚举变量只能从枚举类型中列出的枚举元素中取值。
- 如果试图从枚举类型列出的元素外取值，则会发生错误。

```
num Color myColor=red;          正确
myColor=blue;                   正确
myColor=yellow;                 不正确
```

关于枚举变量赋值的说明

- 说明:

(1)C编译对枚举类型的**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**

关于枚举变量赋值的说明

(2) 每一个枚举元素都代表一个整数，C语言编译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: Weekday workday=mon;

  相当于workday=1;

关于枚举变量赋值的说明

(2) 每一个枚举元素都代表一个整数，C语言编译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,weekend;
```

- ◆指定枚举常量sun的值为7，mon为1，以后顺序加1，sat为6。

关于枚举变量赋值的说明

(3) 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

枚举应用举例

- 实现一个这样的程序：根据用户输入的时间段（0表示上午，1表示下午，2表示晚上）来输出相应时间的“问候”。
- 解题思路：
  - 构造包含“上午”、“下午”、“晚上”三个元素的枚举类型。
  - 设计一个“问候”的函数，把枚举变量作为参数传递进去。

```
1.  #include <stdio.h>
2.  enum time_of_day {Morning, Afternoon, Evening};
3.  void hello(enum time_of_day);

4.  int main()
5.  {
6.      enum time_of_day tod;
7.      puts("请输入时间段（0表示上午，1表示下午，2表示晚上）");
8.      scanf(“%d”, &tod);
9.      hello(tod);
10.     return 0;
11. }
12. void hello(enum time_of_day tod)
13. {
14.     switch (tod)
15.     {
16.         case Morning: puts("Good morning!"); break;
17.         case Afternoon: puts("Good afternoon!"); break;
18.         case Evening: puts("Good evening!"); break;
19.         default: puts("Hello!");
20.     }
21. }
```

什么是typedef

- typedef == type define? ------ No!
- typedef被称为用户自定义类型，但其实并不是用户自己定义的一种新的数据类型，而是用户根据自己的需要给某种数据类型重新命名。
- 一般声明形式：

typedef 类型名 标识名;

◆其中，“类型名”必须是已有定义的类型标识符，“标志名”是用户定义的标识符，作为这个类型的别名

typedef的作用

1.简单地用一个新的类型名代替原有的类型名

```
typedef int Integer;
typedef float  Real;
```

int i,j; float a,b; 与

Integer i,j ; Real a,b; 等价

typedef 的作用

2.命名一个简单的类型名代替复杂的类型表示方法

(1)命名一个新的类型名代表结构体类型：

```
typedef struct
{int month; int day; int year;} Date;
Date birthday;
Date *p;
```

typedef 的作用

2.命名一个简单的类型名代替复杂的类型表示方法

(2) 命名一个新的类型名代表数组类型

```
typedef int Num[100];
Num a;
```

typedef 的作用

2.命名一个简单的类型名代替复杂的类型表示方法

(3)命名一个新的类型名代表一个指针类型

```
typedef char *String;
String p; //表示 char * p;
```

创建typedef语句的简单方法

- 归纳起来，声明一个新的类型名的方法是
  - ① 先按定义变量的方法写出定义体（int i;）
  - ② 将变量名换成新类型名（将i换成Count）
  - ③ 在最前面加typedef
    （typedef int Count）
  - ④ 用新类型名去定义变量（Count i）

创建typedef语句的示例1

- 以定义上述的数组类型为例来说明：

① 先按定义数组变量形式书写：int a[100];

② 将变量名a换成自己命名的类型名：

int Num[100];

③ 在前面加上typedef，得到

typedef int Num[100];

用来定义变量：Num a;

相当于定义了：int a[100];

创建typedef语句的示例2

- 对字符指针类型，也是：

```
char *p;
char *String;
typedef char *String;
String p;
```

关于typedef的其他说明

(1)以上的方法实际上是为特定的类型指定了一个同义字(synonyms)。例如

①typedef int Num[100];

Num a; Num是int [100]的同义词

②typedef char* PCHAR;

PCHAR p1; PCHAR是char*的同义词

关于typedef的其他说明

(2) 用typedef只是对已经存在的类型指定一个新的类型名，而没有创造新的类型。

(3)用typedef声明数组类型、指针类型，结构体类型、共用体类型、枚举类型等，可以简化代码，使得编程更加方便，不易出错，便于理解。

```
typedef struct student * PStudent;
PStudent xiaoming;
```

关于typedef的其他说明

(4) 当不同源文件中用到同一类型数据时，常用typedef声明一些数据类型。可以把所有的typedef名称声明单独放在一个头文件中，然后在需要用到它们的文件中用#include指令把它们包含到文件中。这样编程者就不需要在各文件中自己定义typedef名称了。

关于typedef的其他说明

(5) 使用typedef名称有利于程序的通用与移植。有时程序会依赖于硬件特性，用typedef类型就便于移植。

- 比如，sizeof的数据类型size_t就是一个自定义类型；
- 不同计算机平台sizeof运算符的计算结果可能有不同的结果，因此ANSI C决定交由各平台自行定义。许多计算机将其定义为unsigned int类型，即typedef unsigned int size_t;

作业 2017/12/22

- 按下列要求编写程序，提交手写源代码

1. 一个档案管理程序的对象为教师和学生，每个档案包含如下信息：

   （1）编号；（2）姓名；（3）年龄；（4）分类-教师（用't'表示）或学生（用's'表示）；（5）教师职称；（6）教师教的课程名（三门）；（7）学生入学年份；（8）学生的成绩（三门）

   ① 试用C语言定义一结构体类型Document，声明上述信息，并在适当的地方使用共用体定义。

   ② 假定档案已经存放在名为person[]的数组中，试编写一个函数void printDoc(Document *p, int N)打印输出全部人员档案的信息，其中N为档案总数。