一天的学习和生活开始了…

起床 → 吃早餐 → 上课 → 吃午餐 → 午休 → 上课 → 锻炼 → 晚自习

按顺序

“天而复始”不断循环

按部就班 …

吃午餐 → 如果（  ）→ 

否则 → 午休

→ 上课

选择

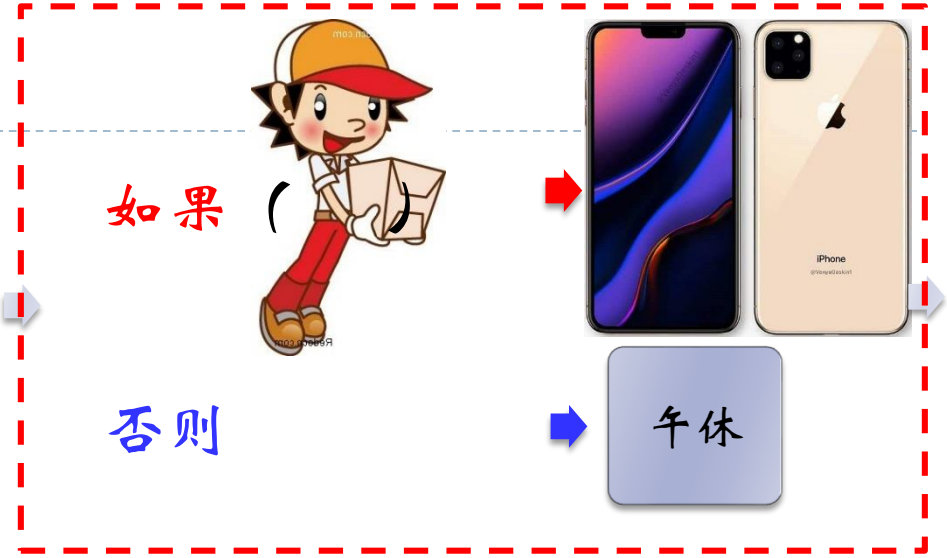
3 程序的流程控制方法 (Flow Control)

郭延文

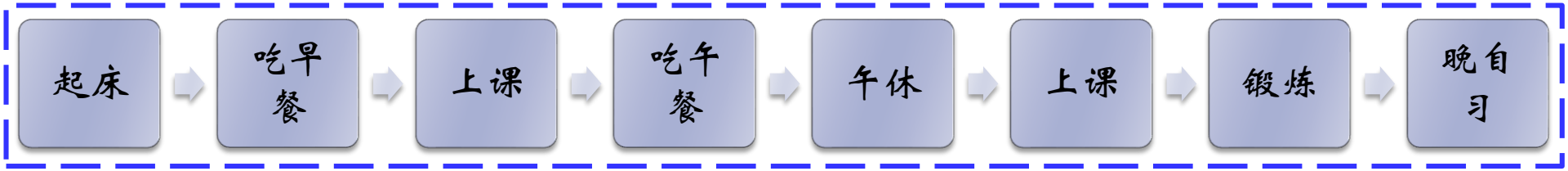
2019级计算机科学与技术系

程序的流程控制方法

- 分支流程控制方法



- 循环流程控制方法



周一到周五 按天重复... 循环！

流 程

- **语句**的执行次序
  - 一般情况下，程序中的语句按照排列次序**顺序**执行；
  - 如果遇到控制语句，比如C语言中的if语句和while语句等，则会改变执行的顺序；

  - 控制语句改变执行次序有**分支**和**循环**两种基本方式。
  - 逻辑上：**顺序**、**分支**和**循环**是程序中的三种基本流程。

  - 一个程序的总流程由基本流程衔接而成。

大家的生活何尝不是这样！

表达式语句

- 在C++表达式的后面加上一个分号“;”就可以构成表达式语句，其格式为：

  <表达式>;

例如：

- a + b * c;
- a > b ? a: b;
- a++;
- x = a | b & c;

- 一个表达式语句执行完后将执行紧接在后面的下一个语句

常用表达式语句

- 赋值
- 自增/自减
- 函数调用
- 输入/输出

例如

- x = a+b; //赋值
- x++; //自增，单独一条语句相当于：x=x+1;
- f(a); //函数调用
- cin >> a; //c++输入；c输入：scanf, getchar
- cout << b; //c++输出；c输出：printf, putchar
- …… …

复合语句

- 复合语句是由一对花括号括起来的一条或多条语句，又称为块（block）。其格式为：

  { <语句序列> }

  - <语句序列>中的语句可以是任何的C++语句，其中包括数据定义和声明语句。

- 复合语句中的语句序列一般按照书写次序执行。
- 语法上，复合语句可看作是一个语句。
- 复合语句一般作为函数体和结构语句（选择和循环）的成分语句。

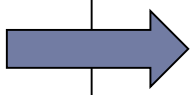
复合语句举例

```
{
    int a,b;
    cin >> a >> b;
    int max;
    max = (a >= b)?a:b;
    cout << max << endl;
}
```

一个完整的例子

- 例0 计算一组圆（直径为n以内的正整数）的周长之和（计量单位为米）。

```
#include <stdio.h>
#define PI 3.14
int main( )
{  int n, d = 1;
   double sum = 0;
   char ch = 'm';
   printf("Input n: ");
   scanf("%d", &n);
   ……
   return 0;
}
```



```
while(d <= n)
{
        sum = sum + PI * d;
        d = d + 1;
}

printf("The sum is: %f ", sum);
putchar(ch);   //显示计量单位
```

上机常见问题 – 一个建议

```
int main ()
{
    while ()
    {
        …
```

不好的书写习惯！

好的书写习惯：先写上复合语句的一对大括号，再写内容（函数体、循环体、其他复合语句）！

```
int main ()
{
        …


}
```

```
while(… )
{
        …


}
```

VC assistant（VC 助手）

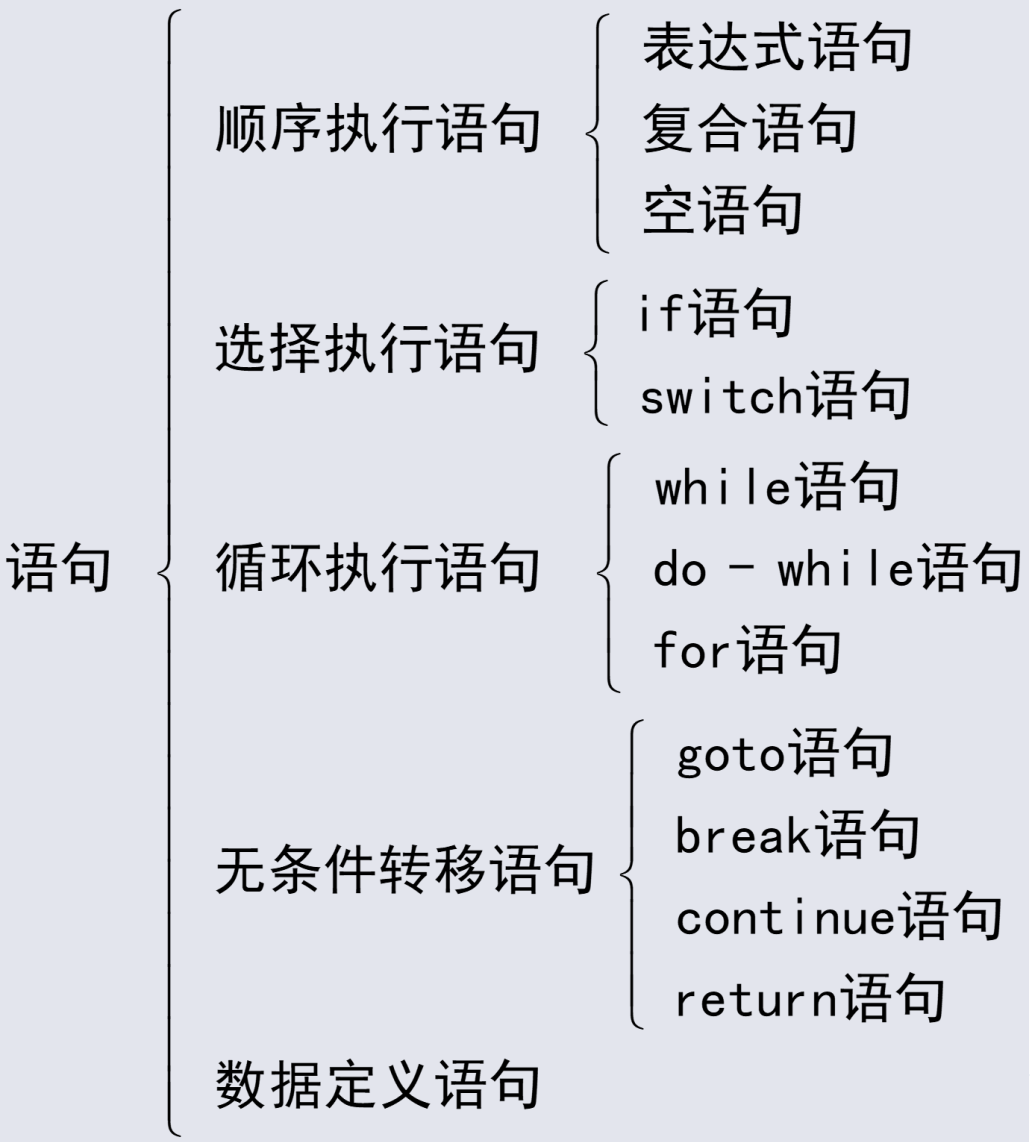
- VC编写程序的辅助工具，包括
  - 语言着色
  - 编写注释
  - 变量提示
  - 函数提示
  - …
- 怎么下载？

  - 自己动手呗！

三种基本流程

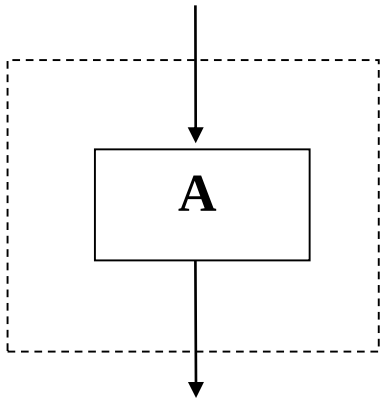
- 顺序
- 分支
- 循环

C/C++语句的分类


语句
顺序执行语句
表达式语句
复合语句
空语句
选择执行语句
if语句
switch语句
循环执行语句
while语句
do - while语句
for语句
无条件转移语句
goto语句
break语句
continue语句
return语句
数据定义语句

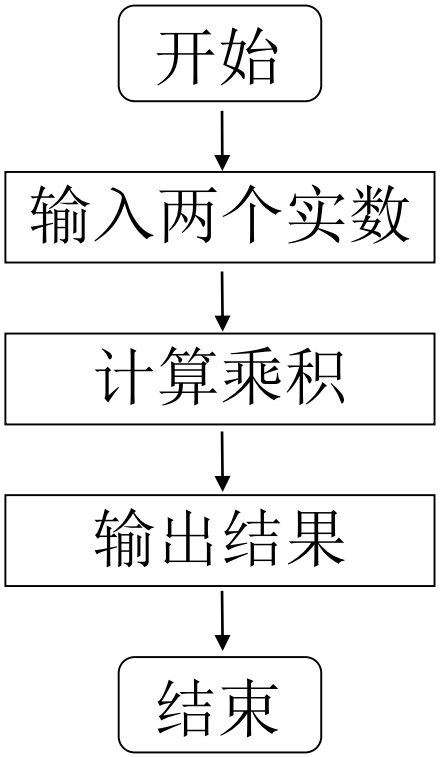
3.1 顺序流程

- 按源程序的语句从上到下逐句执行，每条语句执行一次，是没有控制语句控制的流程



编写程序(Coding)

- 输入两个实数，输出它们的乘积



- 顺序流程中，语句的书写次序就是程序的执行次序。
- 语句相同，只是次序稍有不同，结果可能会输出不同的值：

  - 片段1：

```
int i = 1, sum = 0;
i = i + 1;
sum = sum + i;
printf("sum = %d \n", sum);                    //输出sum = 2
```

  - 片段2：

```
int i = 1, sum = 0;
sum = sum + i;
i = i+1;
printf("sum = %d \n", sum);                    //输出sum = ?
```

3.2 分支流程

- 分支流程的基本形式
- 分支流程的嵌套
- C语言其他分支流程控制语句

分支流程的基本形式

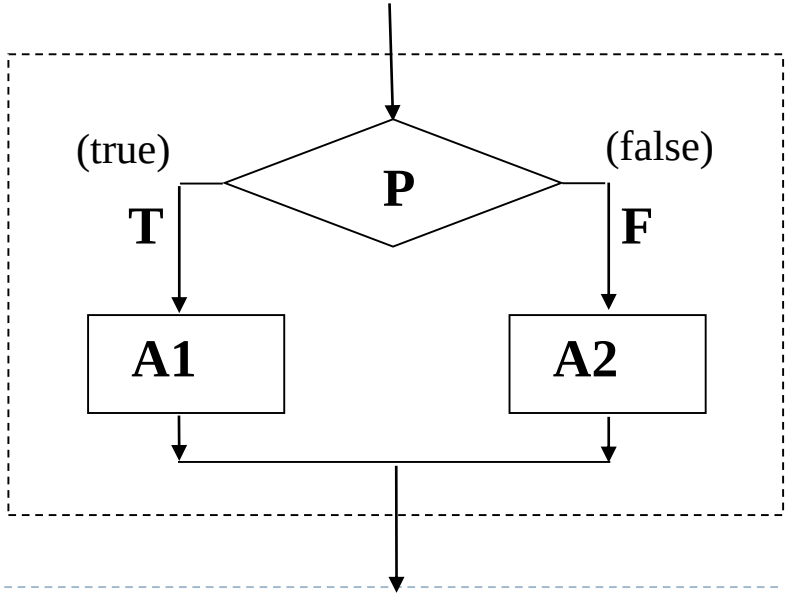
- 分支流程用于选择性计算场合

  当（）　干什么...

  否则　干什么 ...

- 典型的分支流程，包含一个条件判断和两个分支任务

  - 先判断条件P
    - 当条件P成立时，执行任务A1；然后结束该流程；
    - 当条件P不成立时，执行任务A2；然后结束该流程。



分支流程的基本语句 if…else

```
if (<条件P>) // if 和括号间可以加一空格
{
        …
}
else
{
        …
}
```

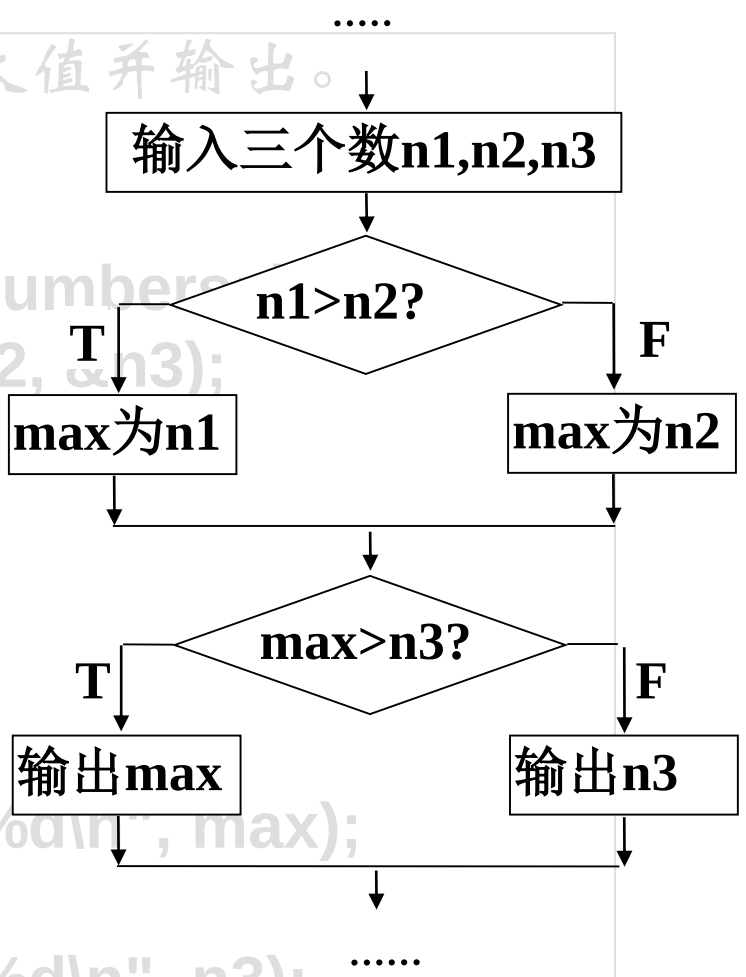
- <条件P>一般是带有关系或逻辑操作的表达式，表达式的值为true，则认为条件成立，执行if子句，否则执行else子句
  - 关系操作：比如 n > 10
  - 逻辑操作：比如 n > 10 && n < 100 (&&：并且)
    n < 10 || n > 100 (||: 或者)

例：输入三个整数，求最大值并输出

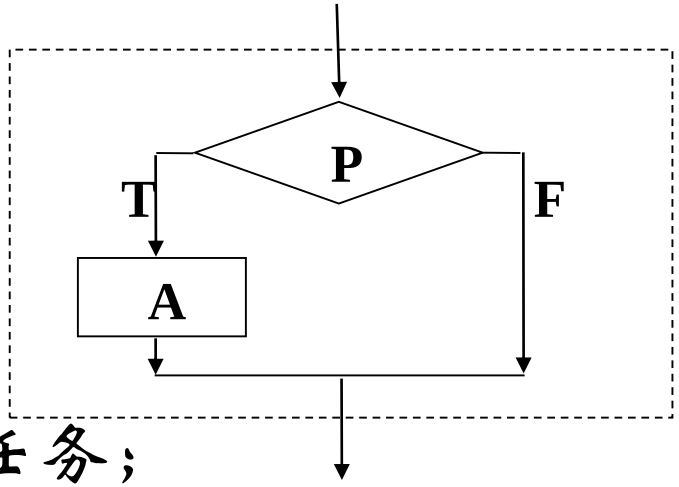
```
#include <stdio.h>
int main()
{       int n1, n2, n3, max;
        printf("Please enter three numbers: \n");
        scanf("%d%d%d", &n1, &n2, &n3);
        if(n1 > n2)
                max = n1;
        else
                max = n2;
        if(max > n3)
                printf("The max. is: %d\n", max);
        else
                printf("The max. is: %d\n", n3);
        return 0;
}
```

例1-4 求输入三个整数中的最大值并输出。

```
#include <stdio.h>
int main()
{
    int n1, n2, n3, max;
    printf("Please enter three numbers:");
    scanf("%d%d%d", &n1, &n2, &n3);
    if(n1 > n2)
        max = n1;
    else
        max = n2;
    if(max > n3)
        printf("The max. is: %d\n", max);
    else
        printf("The max. is: %d\n", n3);
    return 0;
}
```



- 分支流程的另外一种形式，包含一个条件判断和一个分支任务



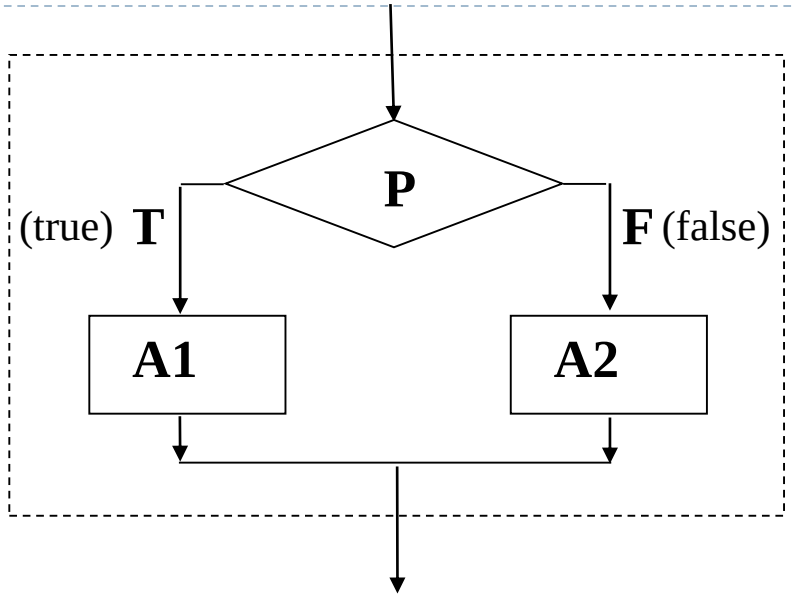
  - 先判断条件P
    - 当条件P成立时，执行任务A；然后结束该流程；
    - 当条件P不成立时，不执行任务；然后结束该流程。

- C语言中的if...语句实现这种分支流程的控制

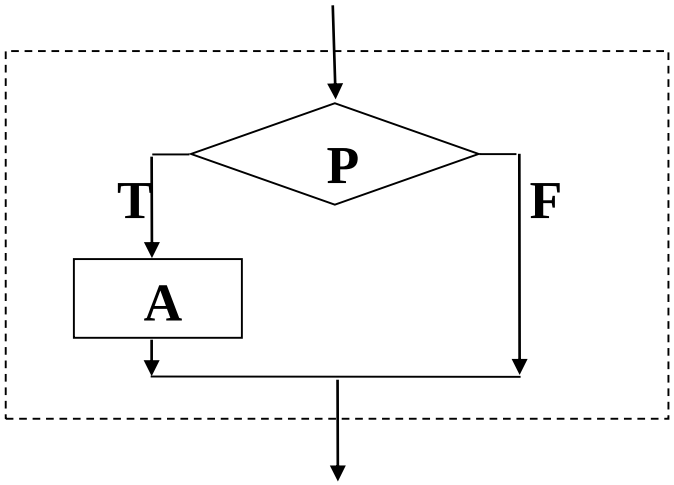
```
if (<条件P>)
{
    …
}
```

  - 条件P一般是带有关系或逻辑操作的表达式，表达式的值为true，则认为条件成立，执行if子句；否则不执行任何子句

分支流程的两种基本形式



if else



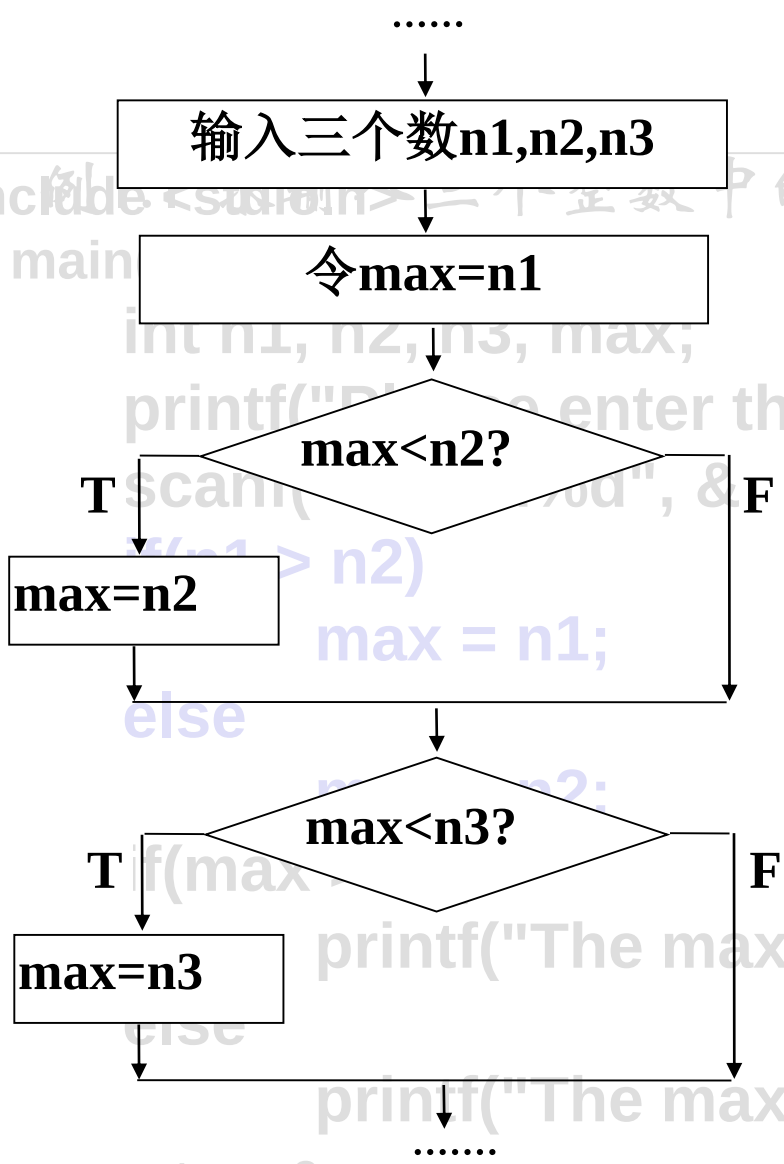
if

- 分支流程中的条件只判断一次，每个任务最多只执行一次

例：输入三个整数，求最大值并输出

```
#include <stdio.h>
int main()
{       int n1, n2, n3, max;
        printf("Please enter three numbers: \n");
        scanf("%d%d%d", &n1, &n2, &n3);
        if(n1 > n2)
                max = n1;
        else
                max = n2;
        if(max > n3)
                printf("The max. is: %d\
        else
                printf("The max. is: %d\n", n3);
        return 0;
}
```

```
max = n1;
if(max < n2)
        max = n2;
if(max < n3)
        max = n3;
//输出max
```




```
max = n1;
if(max < n2)
        max = n2;
if(max < n3)
        max = n3;
//输出max
```

分支流程的书写

- 编写if语句时，采用（tab键）缩进形式
  - 保持前后一致，使程序美观且提高程序的可读性。

```
if(x >= 0)
        y = x * x;
else
        printf("Input error! \n");
```

  - 如果分支任务含多条语句，则一定要用一对花括号将它们组合成复合语句。

```
if(x >= 0)
{
        y = x * x;
        printf("%lf*%lf equal %lf \n", x, x, y);
}       //复合语句是一个整体，要么都被执行，要么都不被执行
else
        printf("Input error! \n");
```

分支流程的嵌套

- 多个分支流程可以嵌套，成为多分支形式。

```
#include <stdio.h>
int main()
{       int n1, n2, n3, max;
        printf("Please enter three nu
        scanf("%d%d%d", &n1, &n2
        if(n1 > n2)
                max = n1;
        else
                max = n2;
        if(max > n3)
                printf("The max. is: %
        else
                printf("The max. is: %
        return 0;
}
```

```
if(n1 > n2)
        if(n1 > n3)
                max = n1;
        else
                max = n3;
else
        if(n2 > n3)
                max = n2;
        else
                max = n3;
//输出max
```

- 在编辑嵌套的if语句时，更应采用结构清晰的缩进格式。不过，如果if语句嵌套层次很深，缩进会使程序正文过分偏右，给程序的编辑、查看带来不便，建议改写：

```
if(score >= 90)
        printf("A \n");
else
        if(score >= 80)
                printf("B \n");
        else
                if(score >= 70)
                        printf("C \n");
                else
                        if(score >= 60)
                                printf("D \n");
                        else
                                printf("Fail \n");
```

```
if(score >= 90)
        printf("A \n");
else if(score >= 80)
        printf( “B \n");
else if(score >= 70)
        printf("C \n");
else if(score >= 60)
        printf("D \n");
else
        printf("Fail \n");
```

if, else if, …, else

```
if (P1)
        do A1;
else if (P2) //不满足P1，满足P2
        do A2;
else if (P3) // 不满足P1, P2， 满足P3，
        do A3;
else          // 不满足P1, P2, P3
        do A4;
```

```
if(score >= 90)
        printf("A \n");
else if(score >= 80)
        printf( “B \n");
else if(score >= 70)
        printf("C \n");
else if(score >= 60)
        printf("D \n");
else
        printf("Fail \n");
```

嵌套的分支流程

- 嵌套的分支流程往往能够避免不必要的条件判断，比如：

```
if (score >= 90)
    printf("优");
……
if (score >= 60 && score<70)
    printf("及格");
if (score < 60)
    printf("不及格");
```

```
if (score >= 90)
    printf("优");
……
else if (score >= 60)
    printf("及格");
else
    printf("不及格");
```

- 改写成嵌套形式，在score >= 90时可以避免后面多个条件判断，例如某个同学95分

例子：从键盘输入3个数字，代表三角形的三条边长，判断其为何种三角形(等边、等腰、直角、其他…)

```
#include <iostream>
using namespace std;
int main()
{     int a,b,c;
      cin >> a >> b >> c;
      if (a+b <= c || b+c <= a || c+a <= b)
            cout << "不是三角形";
      else if (a == b && b == c)
            cout << "等边三角形";
      else if (a == b || b == c || c == a)
            cout << "等腰三角形";
      else if (a*a+b*b == c*c || b*b+c*c == a*a || c*c+a*a == b*b)
            cout << "直角三角形（非等腰）";
      else
            cout << "其它三角形";
      cout << endl;
      return 0;
}
```

对特殊情况的处理可能影响、甚至决定程序的Robustness

- C程序中，当两种不同形式的if语句嵌套时，理解时会产生分歧。

```
max = n1;
if(n1 > n2)
        if(n3 > n1)
                max = n3;
else            //else对应n1>n2还是n3 > n1不成立的情况？
……
```

- 缩进并不改变程序的逻辑。
- else子句与上面最近的if子句(n3>n1)配对，而不是和较远那个if子句(n1>n2)配对！

- 如果在逻辑上需要将else子句与较远的if子句配对
  - 可以用一个花括号把较近的if子句写成复合语句

```
max = n1;
if(n1 > n2)
{     if(n3 > n1)
            max = n3;
}
else                              /*这里else是对应n1 > n2不成立的情况*/
```

  - 或者在较近的if子句后面构造一个用else和分号构造一个分支

```
max = n1;
if(n1 > n2)
      if(n3 > n1)
            max = n3;
      else                        /*这里else是对应n3 > n1不成立的情况*/
            ;
else                              /*这里else是对应n1 > n2不成立的情况*/
```


国 庆 佳 节 普 天 同 庆
欢度国庆
热烈庆祝中华人民共和国成立70周年

内容回顾

良好的编程习惯：从第一步做起！

好的程序：
- 正确
- 可靠
- 高效
- 易读
- 可重用
- 可移植

……

现阶段

确保程序的正确性

- 选择合适的方法（算法）
- 成功运行、（验证多个测试用例）得到正确结果

提高程序的易读性

- 工程名的命名(源.cpp?)
- 自定义标识符的命名
  - 匈牙利命名法
- 注意代码排版
- 适当的注释

本课程自定义标识符命名具体建议☆

- 【总则】采用一致的、有意义的标识符名字。
- 【建议1】自定义标识符应直观、用词准确，可望文知意。切忌使用汉语拼音来命名！
- 【建议2】标识符的长度应当符合min-length && max-information原则。常见的i, j, k, m, n, x, y, z等，可用作局部变量。
- 【建议3】程序中不要出现仅靠大小写区分的相似的标识符。
- 【建议4】用一对反义词命名具有相反含义的变量或函数等。例如：

```
int   minValue，maxValue;
```

本课程自定义标识符命名具体建议☆

- 【建议5】 函数名和类型名用大写字母开头的单词组合而成。如：

```
void  Init(void);
void  SetValue(int value);
```

- 【建议6】 变量名和参数名的首单词用小写字母开头，且使用前缀（参考匈牙利命名法）。如：

```
int nFlag;
int nStuAge;
int nCurrent_value
```

- 【建议7】习惯使用符号常量，符号常量名全用大写字母，用下划线分割单词。如：

```
#define MAX_LENGTH  100
#define PI 3.14
```

本课程自定义标识符命名具体建议☆

前缀	类型	例子
n	int	nLength
c	char	cGrade
f	float	fScore
a	数组	aStu
p	指针	pFunc
C	类或者结构体	CDocument，CPrintInfo
g_	全局变量	g_Servers
m_	成员变量	m_pDoc，m_nCustomers

内容回顾

一天的学习和生活开始了…

起床 → 吃早餐 → 上课 → 吃午餐 → 午休 → 上课 → 锻炼 → 晚自习

按顺序

“天而复始”
不断循环

按部就班 …

吃午餐 → 如果（  ）→ 

否则 → 午休

→ 上课

选择

内容回顾

- 三种基本流程：顺序、分支、循环

- 分支流程

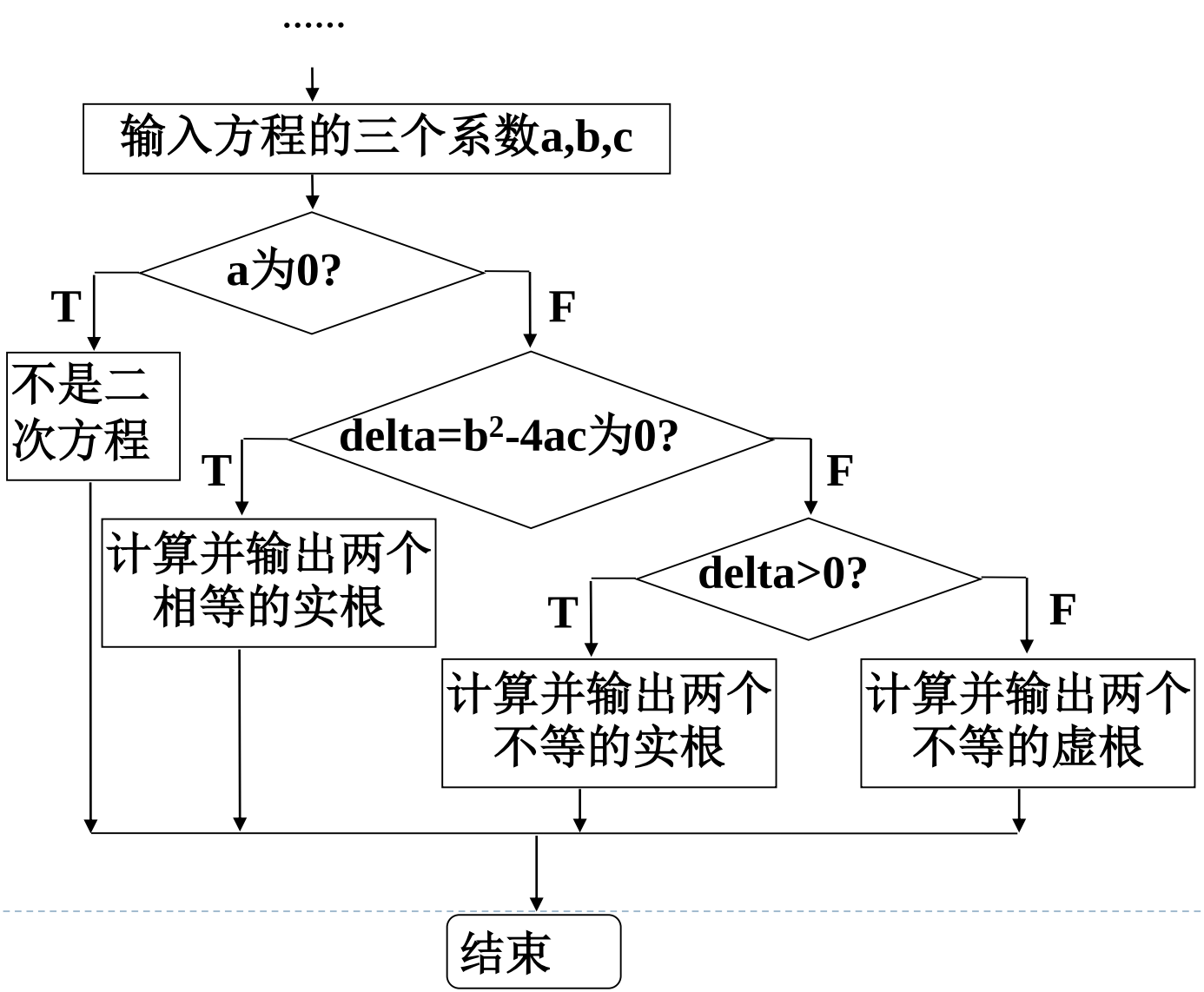
```
if (<条件P>)
{
   …
}
else
{
   …
}
```

```
if (<条件P>)
{
   …
}
```

```
if (<条件P>)
{
   …
}
else if (<条件P1>)
{
   …
}
…
else
{
   …
}
```

例：用求跟公式求一元二次方程$ax^2+bx+c=0$的根

例：用求跟公式求一元二次方程$ax^2+bx+c=0$的根



sqrt()
pow(x, y)

```
…
#include <math.h>
int main( )
{       double a, b, c, delta, p, q;
        printf("Please input three coefficients of \
   the equation: \n");      //上一行续行符后面不能有注释，本行之前没有空格。
        scanf("%lf%lf%lf", &a, &b, &c);
        if(a == 0)          // 这里切勿写成if(a = 0)
                printf("It is not a quadratic equation! \n");
        else if((delta = b*b - 4*a*c) == 0)
                printf("x1 = x2 = %.2f \n", -b / (2 * a));
        else if (delta > 0)
        {       p = -b / (2*a);
                q = sqrt(delta) / (2*a);
                printf("x1 = %.2f,  x2 = %.2f \n", p + q, p - q);
        }
```

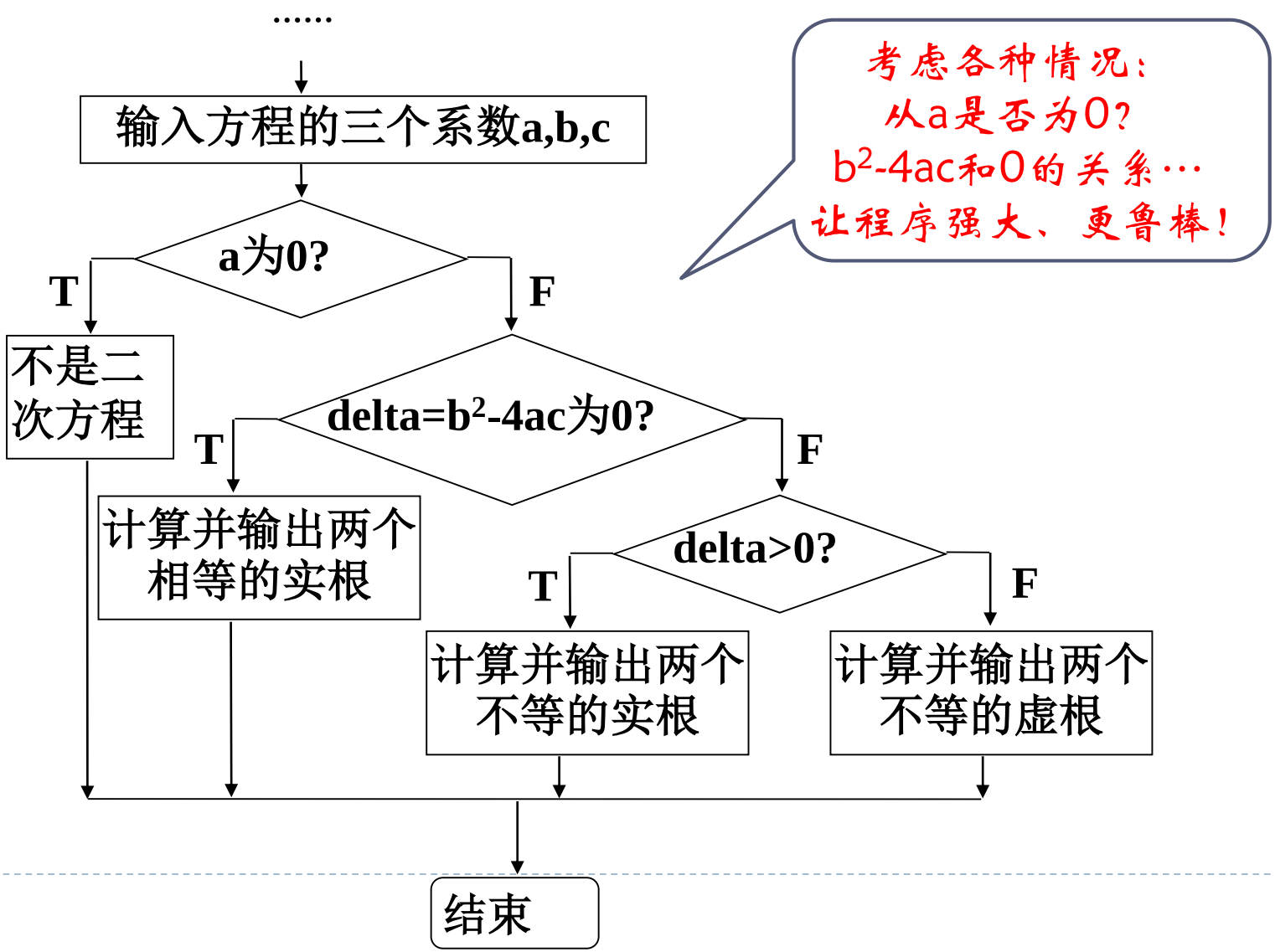
```
        else // < 0 省略不写
        {
                p = -b / (2 * a);
                q = sqrt(-delta) / (2 * a);
                printf("x1 = %.2f + %.2fi,  x2 = %.2f - %.2fi \n", p, q, p, q);
        }
        return 0;
}
```

在输出两个复数根时，采用先分别输出实部和虚部的方法，再添加+、-、i字符来表示复数。

```
Please input three coefficients of the equation:
1.2
2.1
3.4
x1 = -0.88+1.44i,  x2 = -0.88-1.44i
```

例：用求跟公式求一元二次方程$ax^2+bx+c=0$的根

再回顾



一周7天，我们需要根据今天是周几，选择上什么课，做什么事情，如何实现？

例：从键盘输入一个星期的某一天（0：星期天；1：星期一；...），然后输出其对应的英语单词

```
#include <iostream>
using namespace std;
int main()
{ int day;
  cin >> day;
  switch (day)
  {    case 0: cout << "Sunday"; break;
       case 1: cout << "Monday"; break;
       case 2: cout << "Tuesday"; break;
       case 3: cout << "Wednesday"; break;
       case 4: cout << "Thursday"; break;
       case 5: cout << "Friday"; break;
       case 6: cout << "Saturday"; break;
       default: cout << "Input error";
  }
  cout << endl;
  return 0;
}
```

其他分支流程控制语句 – switch语句

- C语言还提供了一种叫做开关语句的switch语句，能实现部分多分支流程的控制。

```
switch(grade)    //该行没有分号
{
        case 'A': printf("90-100 \n");  break;
        case 'B': printf("80-89 \n");   break;
        case 'C': printf("70-79 \n");   break;
        case 'D': printf("60-69 \n");   break;
        case 'F':  printf("0-59 \n");    break;
        default: printf("error \n");
}                //该行没有分号
```

```
char grade;
grade = getchar();
```

switch语句

- switch后面圆括号中的操作结果，与case后面的整数或字符常量进行匹配
  - 如果匹配成功，则从冒号后的语句开始执行，执行到右花括号结束该流程；
  - 如果没有匹配的值，则执行default后面的语句或不执行任何语句（default分支可省略），然后结束该流程。

```
switch(week)     //该行没有分号
{
        case 0: printf("Sunday \n");   break;
        case 1:  printf( “Monday \n"); break;
        ……
        default: printf("error \n");
}                       //该行没有分号
```

```
switch(grade)    //该行没有分号
{
        case 'A': printf("90-100 \n");  break;
        case 'B': printf("80-89 \n");   break;
        case 'C': printf("70-79 \n");   break;
        case 'D': printf("60-69 \n");   break;
        case 'F':  printf("0-59 \n");   break;
        default: printf("error \n");
}                   //该行没有分号
```

```
if(grade == 'A')
        printf("90-100 \n");
else if(grade == 'B')
        printf("80-89 \n");
else if(grade == 'C')
        printf("70-79 \n");
…
else
        printf("error \n");
```

对于能够使用switch语句控制的多分支流程，使用switch语句往往便于编译器进行优化，以便获得比if语句更高的效率。

运用switch语句的注意事项：

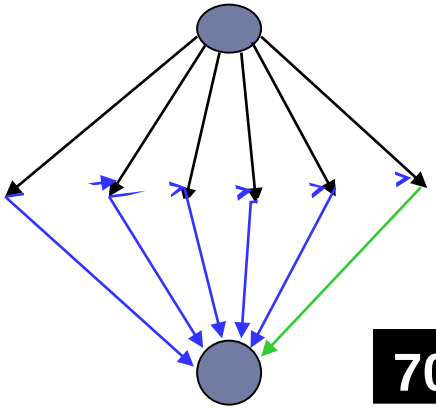
- 每个case后面的语句最多执行一次
- switch后面圆括号中的操作结果与case后面的内容必须保证为一个整数或一个字符
- 须保证case后面的值各不相同，否则无法进行匹配。

```
switch(week)     //该行没有分号
{
        case 0: printf("Sunday \n");   break;
        case 1: printf( “Monday \n");  break;
        ……
        default: printf("error \n");
}                       //该行没有分号
```

- switch语句中的case仅仅与紧随其后的整数或字符常量一起作为语句标号，没有流程控制作用
- “break;”语句具有流程控制作用，它将流程转向switch 语句结束处
- 没有“break;”则执行完本分支任务后，会继续执行其他分支的任务，不管其他分支任务前case后的整数或字符常量是否匹配

C

```
switch(grade)
{
        case 'A': printf("90-100 \n");      break;
        case 'B': printf("80-89 \n");       break;
        case 'C': printf("70-79 \n");       break;
        case 'D': printf("60-69 \n");       break;
        case 'F': printf("0-59 \n");        break;
        default:  printf("error \n");       break;
}
```



70-79

switch语句可以嵌套

- 这时，内层switch语句里的“break;”语句只能将程序的流程转向内层switch语句的结束处，不能控制外层switch语句的流程

```
switch(x)
{
        case 0:  printf("xy = 0 \n"); break;          // 外层分支
        case 1:
                 switch(y)
                 {
                          case 0: printf("xy = 0 \n"); break;// 内层分支
                          case 1: printf("xy = 1 \n"); break; // 内层分支
                          default: printf("xy = %d \n",y);  // 内层分支
                 }
                 break;                                 // 外层分支
        default: printf("error! \n");                   // 外层分支
}
```

C语言其他分支流程控制语句 – goto语句

- C语言还保留了goto语句。goto语句与if语句以及后面某个语句的标号配合使用也能实现分支流程的控制

```
max = n1;
if (n2 < max)
      goto T1
max = n2;
T1:   if (max < n3)
      max = n3;
```

- 但这类流程完全可以不用goto语句，只要改写goto语句所在的if语句条件即可，比如上面的程序片段可以改写成：

```
max = n1;
if (max < n2)
            max = n2;
if (max < n3)
            max = n3;
```

- 由于goto语句容易使程序流程混乱，尽量不要用goto语句实现分支流程的控制！

小结

- 分支流程及其控制方法
  - if...
  - if...else...
  - switch... (break)
  - 嵌套

- 要求：
  - 会运用分支流程控制语句实现简单的分支计算任务

    在main函数中完成数据定义、输入、分支处理、输出
  - 能够定位出（语法）错误行并修改
  - 继续保持良好的编程习惯

上机实验的问题

```
#include <stdio.h>
int main()
{
    int n1, n2, n3, max;
    printf("Please enter three numbers: \n");
    scanf("%d%d%d", &n1, &n2, &n3);
    if(n1 > n2)
        max = n1;
    else
        max = n2;
    if(max > n3)
    {
        printf("The max. is: %d\n", max);
        printf( “ We got it !");
    }
    else
    {
        printf("The max. is: %d\n", n3);
        printf( “ We got it !");
    }
    return 0;
}
```

1. 大括号！
2. 程序排版
   空行
   Tab对齐！
   空格

注：由于ppt的问题，
本课件程序版式（空格多少等）
未必规范！
请大家参考教材的习题
例如：例3-16等

上机实验的问题

```
while (x3 >= 3)
{
        if (n > 2 * m / 3)
            cout << "如果…" << endl,  cout<<endl, x1 = x3 / 3, x2 = x3 / 3,
x3 = x3 - (x1 + x2), i = i + 1;
        else if (n < m / 3)
            cout << "如果..." << endl,  cout <<endl,  x1 = x2 / 3, x3 = x2 - 2
* x1, x2 = x2 / 3, i = i + 1;
        …
}
```

- 逗号连接表达式，符合语法，但一般不这么写；每条有意义的表达式加";"构成语句！
- 空格多了些...关键的运算符前后可以加空格

Q & A