清华大学
Tsinghua University

程序设计基础

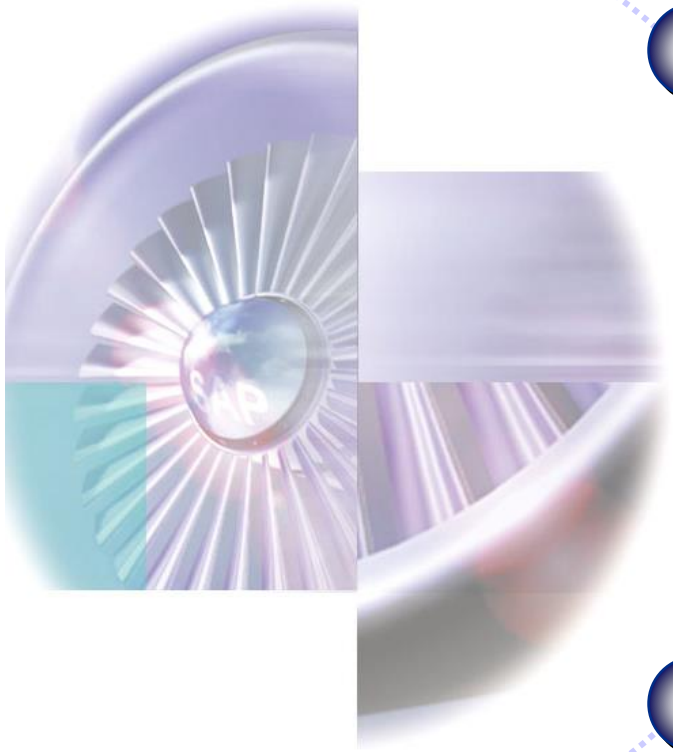
Fundamental of Programming

清华大学软件学院

刘玉身

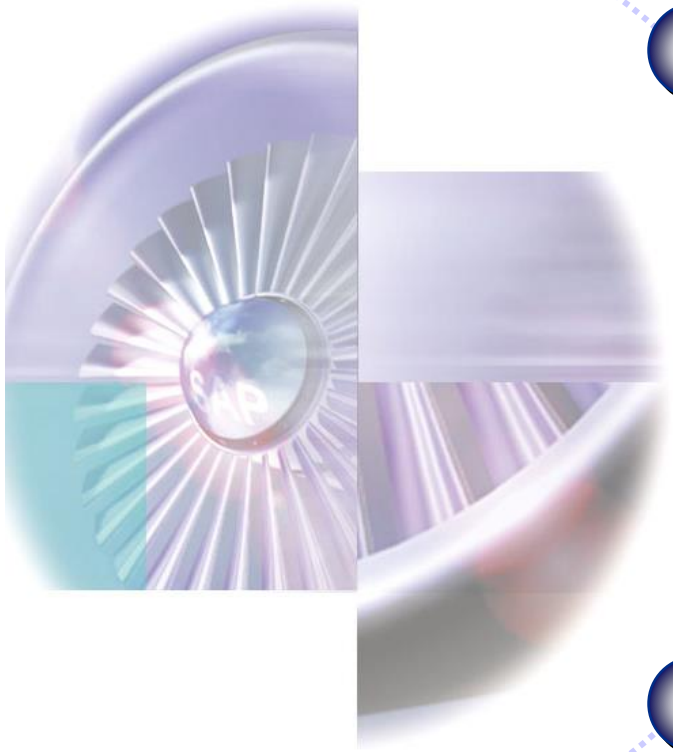
liuyushen@tsinghua.edu.cn

Lecture 6: 函数



1. 函数概述
2. 函数的使用
3. 函数调用的实现过程
4. 函数与设计

Lecture 6: 函数

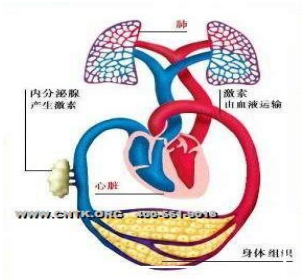


1. 函数概述
2. 函数的使用
3. 函数调用的实现过程
4. 函数与设计

函数概述

顺序

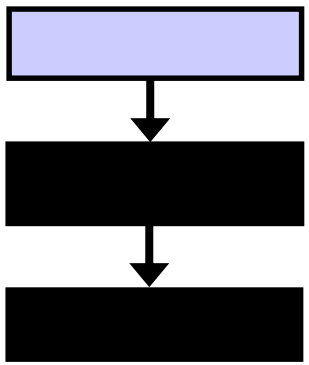
选择

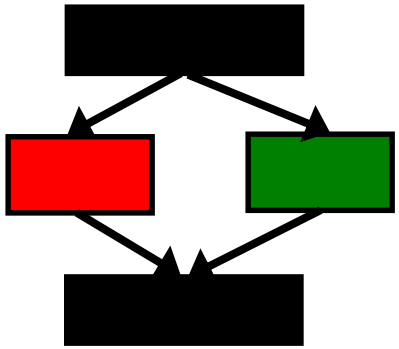


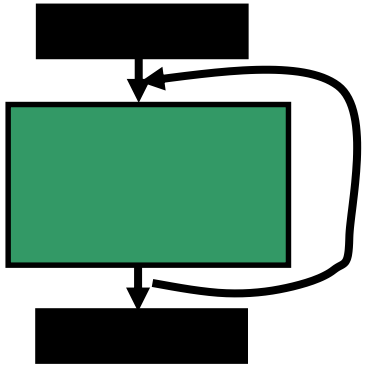
循环

两个特点？

1. 重复执行
2. 连续执行







大学生小明的一周

一周的学习/生活总结：

1. 睡觉7次；
2. 去食堂吃饭21次（或14次）；
3. 上课11次；
4. 去教室上自习6次；
5. 参加拔河比赛1次；
6. 去大礼堂看电影1次；
7. ……

重复不连续！

1. 睡觉；
2. 吃饭；
3. 上课；
4. 上自习；
5. 睡觉；
6. 吃饭；
7. 上课；
8. ……

```
for(i=1; i<=7; i++)
{
    睡觉;
}
for(i=1; i<=21; i++)
{
    吃饭;
}
for(i=1; i<=11; i++)
{
    上课;
}
```

如何处理“重复但不连续”的操作?

——函数

什么是函数？

- 数学里面的函数：正弦函数、余弦函数、指数函数等等；
- C语言里面的函数：也称为子程序（subroutine），它是一组程序代码（包括数据和指令），用来完成某个特定的功能。
- C语言的函数既可以有返回值，也可以无返回值，不像Pascal等语言，严格地区分为函数（function）和过程（procedure）。

C语言的函数类型

- 主函数**main**，每个程序有且仅有一个；
- 库函数，也叫标准函数，由系统提供，用户可以直接使用；
- 自定义函数，用来完成用户特制的功能。

Why函数？

提高工作效率，减少重复劳动！

生活中亦如此...

- 人们常把“一次定义，多次使用”的数据抽象为符号常量（即：宏定义 #define）；
- 同样，人们把“一次定义，多次使用”的行为（或功能）抽象为函数。
- 优点：
  - ✓ 易于理解
  - ✓ 利于程序的修改和升级
  - ✓ 模块化程序设计

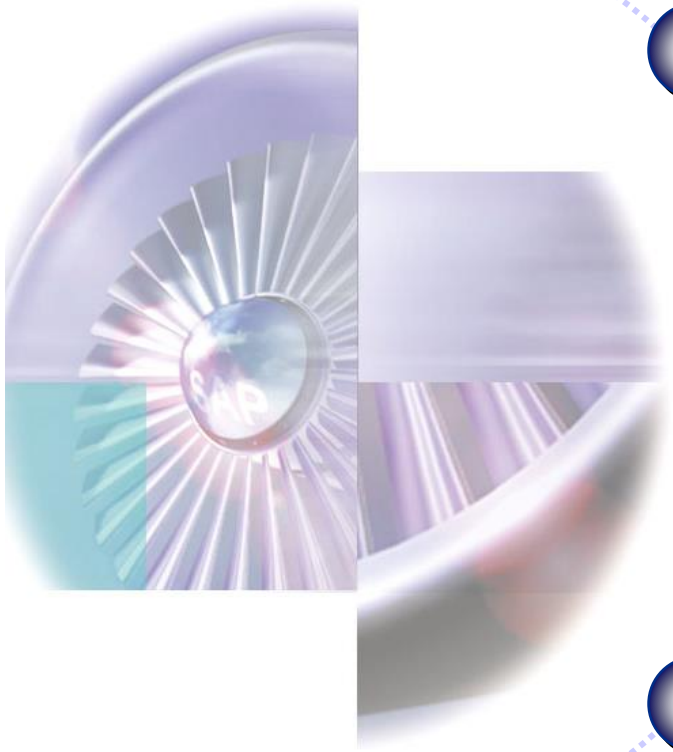
模块化程序设计

- 基本思想：
  - 将一个大的程序按功能分割成一些小模块
- 特点：
  - 各模块相对独立、功能单一、结构清晰、接口简单
  - 控制了程序设计的复杂性
  - 提高了原件的可靠性
  - 缩短开发周期
  - 避免程序开发的重复劳动
  - 易于维护和功能扩充
  - ……

在编程时...

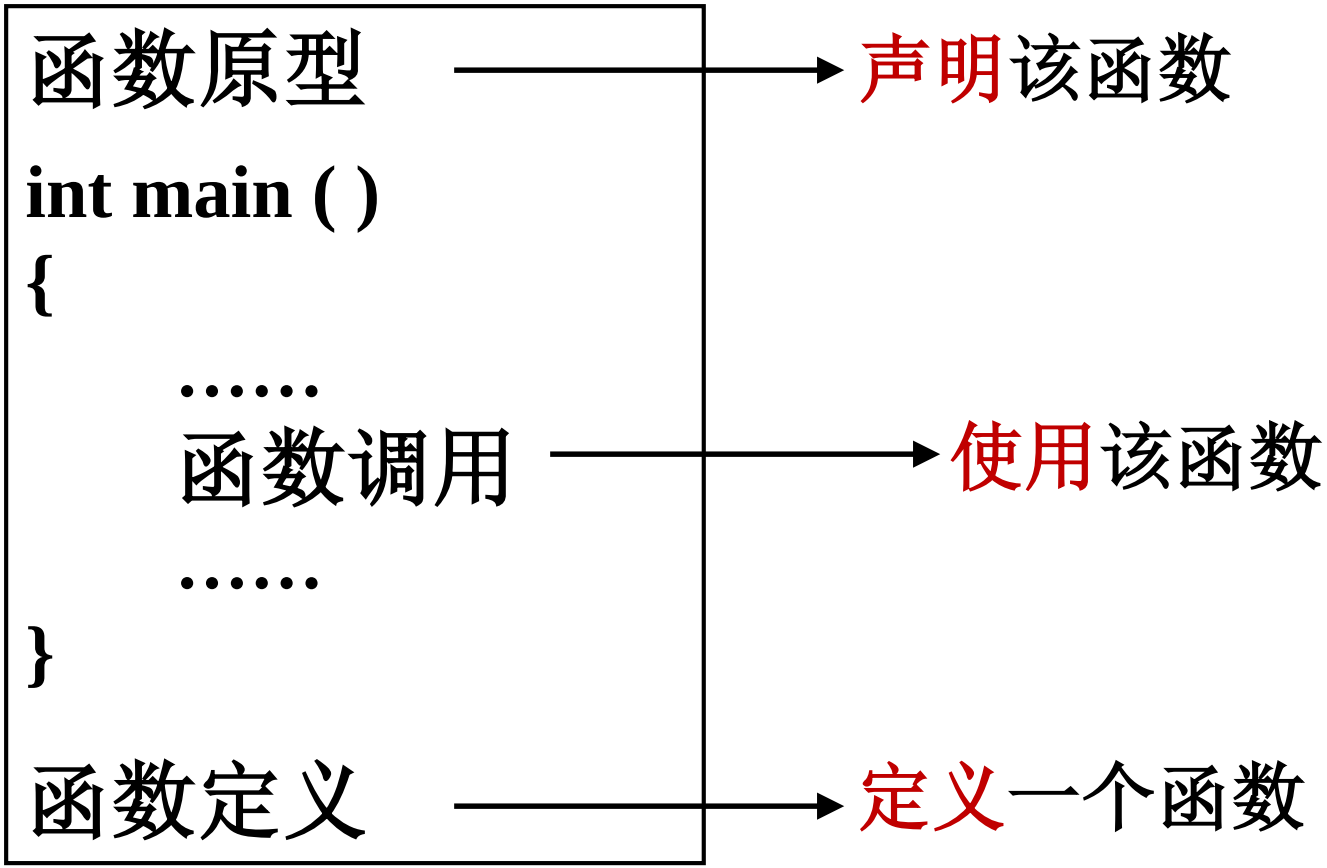
我想从键盘读入一个数据	用scanf函数！
我想生成一个随机数	用rand函数！
我想计算一个字符串的长度	用strlen函数！
我想知道一个人的星座和属相	自己编写函数！

Lecture 6: 函数



1. 函数概述
2. **函数的使用**
3. 函数调用的实现过程
4. 函数与设计

函数的使用



函数的使用模式

函数的定义

函数定义的一般形式：

```
<数据类型>  <函数名>  (<参数列表>)
{
        <数据声明部分>

        <执行语句>
}
```

```
int WhoWin(char child1, char child2)
{
    int result;

    if(child1 == child2)  result = 0;
    else if(child1 == 'S' && child2 == 'J')
        result = 1;
    else if(child1 == 'S' && child2 == 'B')
        result = -1;
    else if(child1 == 'J' && child2 == 'B')
        result = 1;
    ....

    return result;
}
```

函数无返回值时能否用return语句？

return语句是否必须在函数末尾？

在函数中能否出现多个return语句？

函数的声明

在使用一个函数的时候，除了要对它进行定义以外，还要对它进行声明（**function declarations**），即：

给出这个函数的函数原型（**prototype**）。

- 一些函数原型的例子：
  - **void Useless(void);**
  - **void PrintInteger(int value);**
  - **double CalculateTax(double amount, double rate);**
  - **double CalculateTax(double, double);**

函数的使用

1. 用户自定义的函数

```
void hello( );    // 先声明后使用
int main( )
{
    hello( );     // 函数调用
    return 0;
}
void hello( )     // 函数定义
{
    printf("hello");
}
```

分号

2. 使用其他文件中的函数

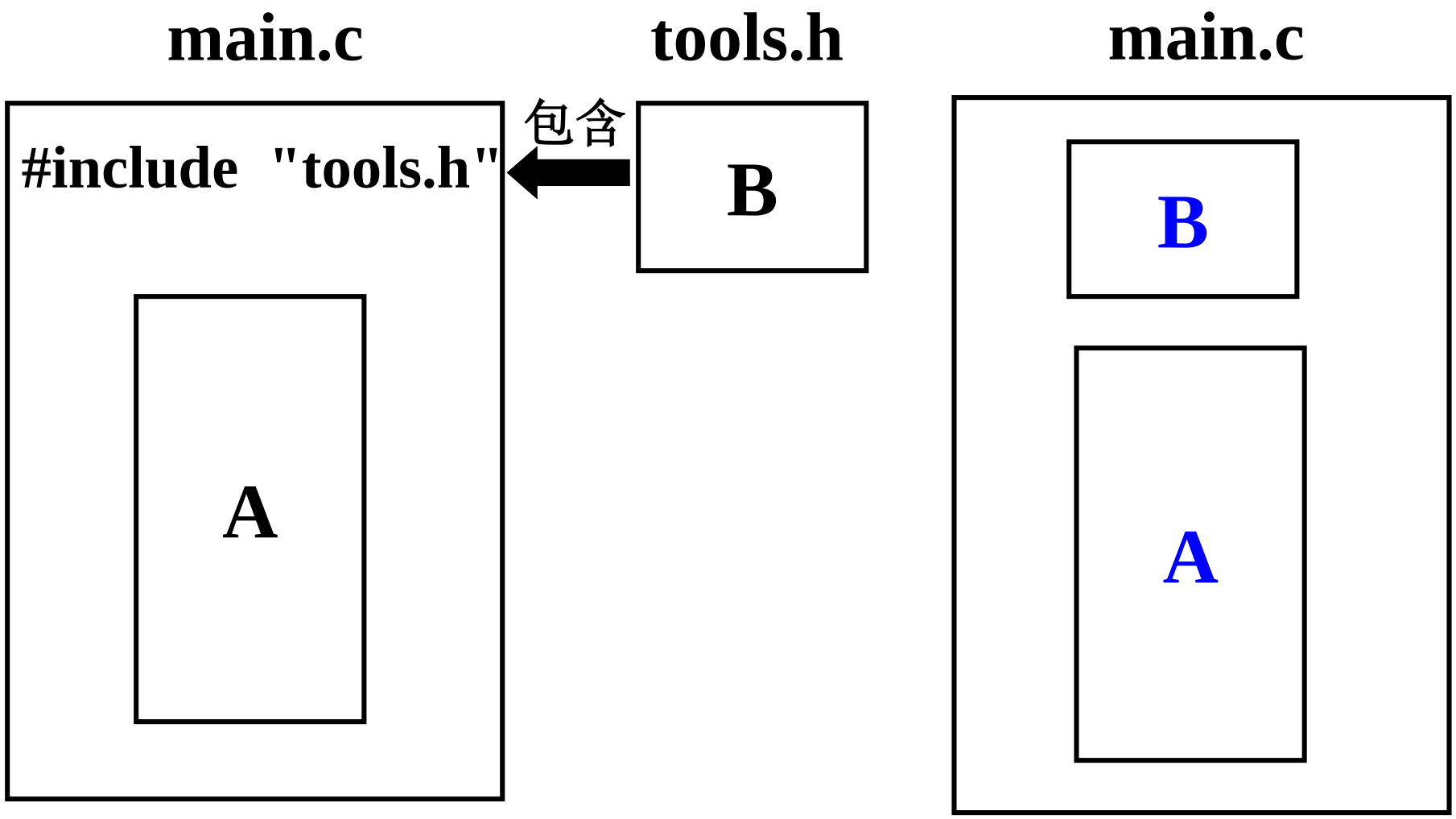
问题描述：

在一个程序中，有多个源文件，如何在一个文件中调用其他文件中定义的函数？

- 假设有源文件**tools.c**，存放一些工具函数
- 另定义一个头文件**tools.h**，存放函数原型
- 在**main.c**中 **#include "tools.h"**
- 调用相应的函数

“文件包含”处理

“文件包含”处理是指一个源文件可以将另外一个源文件的全部内容包含进来。C语言提供**#include** 命令来实现“文件包含”操作。


main.c
tools.h
main.c
#include "tools.h"
包含
B
A
B
A

3. 使用系统提供的库函数

问题描述：

如何在源程序中，调用系统提供的库函数，如printf，scanf等，编译器需要它们的函数原型和代码吗？

- 编译器需要，但程序员不需要；
- 将相应头文件包含进来，如#include <stdio.h>;
- 链接程序知道这些库函数的代码所在的位置，会自动地链接到目标程序当中。

stdio.h

```
.....
_CRTIMP   int __cdecl   printf(const char *, ...);
_CRTIMP   int __cdecl   putc(int, FILE *);
_CRTIMP   int __cdecl   putchar(int);
_CRTIMP   int __cdecl   puts(const char *);
.......
_CRTIMP   int __cdecl   scanf(const char *, ...);
```

#include 命令的两种用法：

- **#include <文件名>：系统到存放C库函数头文件所在的目录中寻找需要包含的文件，这称为标准方式（可在VC中设置）；**
- **#include "文件名"：系统先在用户当前工作目录中寻找要包含的文件，若找不到，再按标准方式查找（即按尖括号方式查找）。**

函数的调用

1. 函数调用的一般形式

函数名（参数1，参数2，…，参数n);

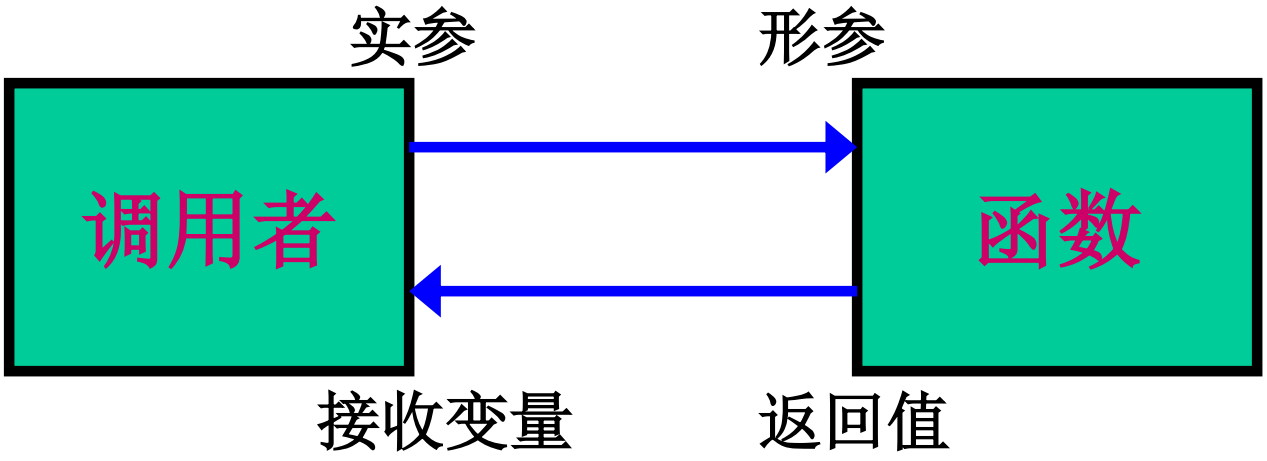
例如：

```
y = sqrt(x);
printf("hello world!");
```

2. 实参与形参

- ☺ 实参：在**调用**一个函数时，所指定的参数称为“实际参数”，可以是常量、变量或表达式；
- ☺ 形参：在**定义**一个函数时，所指定的参数称为“形式参数”，必须是一个变量。

1. 个数？
2. 方向？



示例

actual argument
实参

formal argument
形参

局部变量

```
#include <stdio.h>
int power(int m, int n);
/* test power function */
int main()
{
    int i, x=2;

    for (i=0; i < 10; ++i)
        printf("%d %d %d\n", x, i, power(x,i));
    return 0;
}
/* power: raise base to n-th power */
int power(int m, int n)
{
    int i, p=1;
    for (i = 1; i<=n; ++i)
        p = p * m;
    return p;
}
```

```
int main( )
{
     int salary, nCars, nHouses;

     salary = 3000;
     nCars = 0;
     nHouses = 0;
     DayDreaming(salary, nCars, nHouses);
     printf("%d %d %d", salary, nCars, nHouses);
    return 0;
}
void DayDreaming(int salary, int cars, int houses)
{
     salary = salary * 3;
     cars += 2;
     houses ++;
}
```

实参与形参——补充说明

- 形参：定义函数时函数名后面括号中的变量名
- 实参：调用函数时函数名后面括号中的表达式
- 说明：
  - 实参必须有确定的值（**value is copied** ）
  - 形参必须指定类型
  - 形参与实参类型一致，个数相同
  - 若形参与实参类型不一致，自动按形参类型转换（函数调用转换）
  - 形参在函数被调用前不占内存；函数调用时为形参分配内存；调用结束，内存释放

变量的作用范围

变量的作用范围：也称为变量的作用域，指程序中的一段代码范围，在此范围内，这个变量是有效的，可以被访问。而在此范围之外，该变量是无效的，不能被访问。

一个变量在它的作用域以内是“可见”的，在它的作用域以外是“不可见”的。

变量的作用范围可以分为两类：

- 函数一级的作用范围，即局部变量；
- 文件一级的作用范围，即全局变量。

1. 局部变量

局部变量：在一个函数内部定义的变量

- 局部变量只在本函数范围内有效；
- 在不同函数中可使用相同名字的局部变量；
- 形参也是局部变量，也只能在本函数中使用；
- 局部变量的生存期：当函数被调用时，其局部变量才被创建，并分配相应内存空间；当函数调用结束后，局部变量即消亡，其空间被释放。

```
float  f1(int  a)
{
      int   b,  c;
      …
}
char  f2(int  x,   int  y)
{
      int   i,  j;
      …
}
int  main( )
{
      int   m,  n;
      …
}
```

a、b、c有效

x、y、i、j有效

m、n有效

2. 全局变量

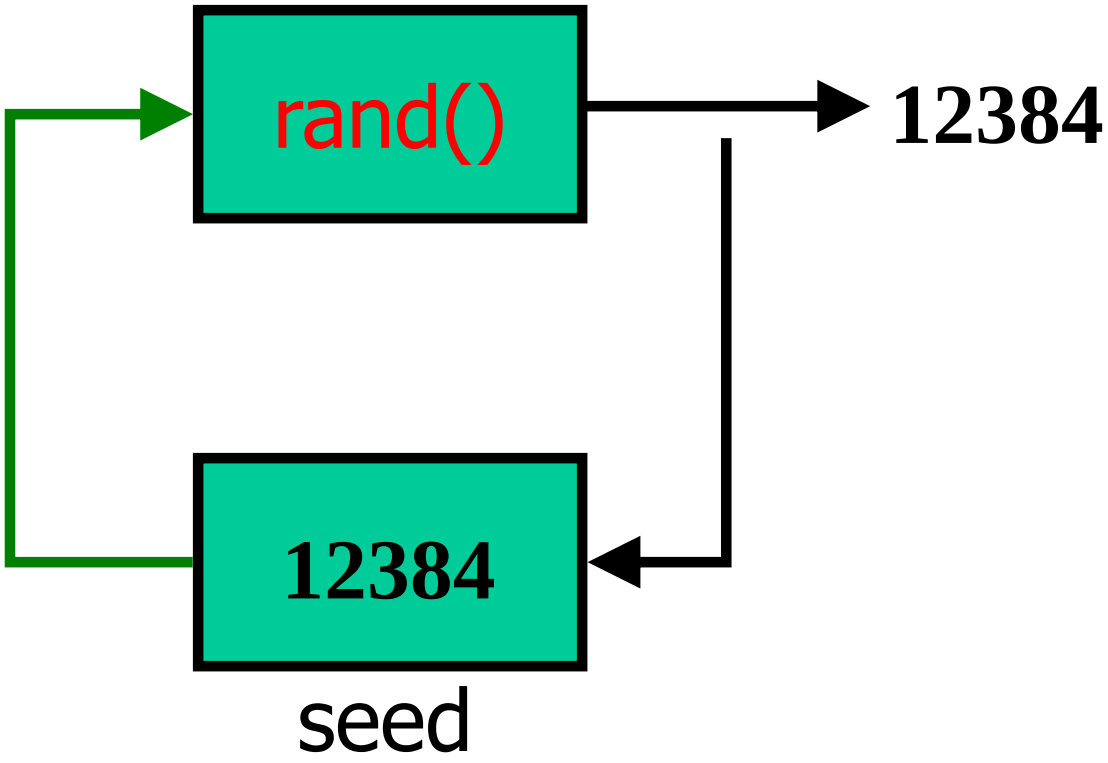
全局变量：在所有函数之外定义的变量

- 全局变量可以为本文件中的其他函数所共用；
- 其有效范围为从定义该变量的位置开始，到本源文件结束为止。

```
int  p, q;
float f1(int a)
{
      int  b, c;
      …
}
char  c1, c2;
char f2(int x,  int y)
{
      int  i, j;
      …
}
int main( )
{
      int  m, n;
      …
}
```

全局变量 c1、c2的 作用范围

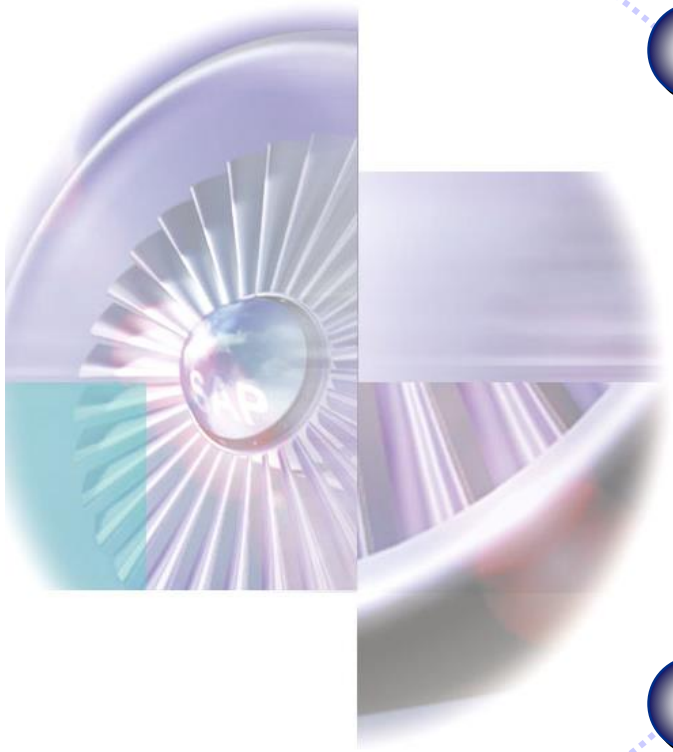
全局变量 p、q的 作用范围


rand()
12384
12384
seed

rand.c

```
int seed;
int rand( int seed )
{
    int seed;
    ......
}
```

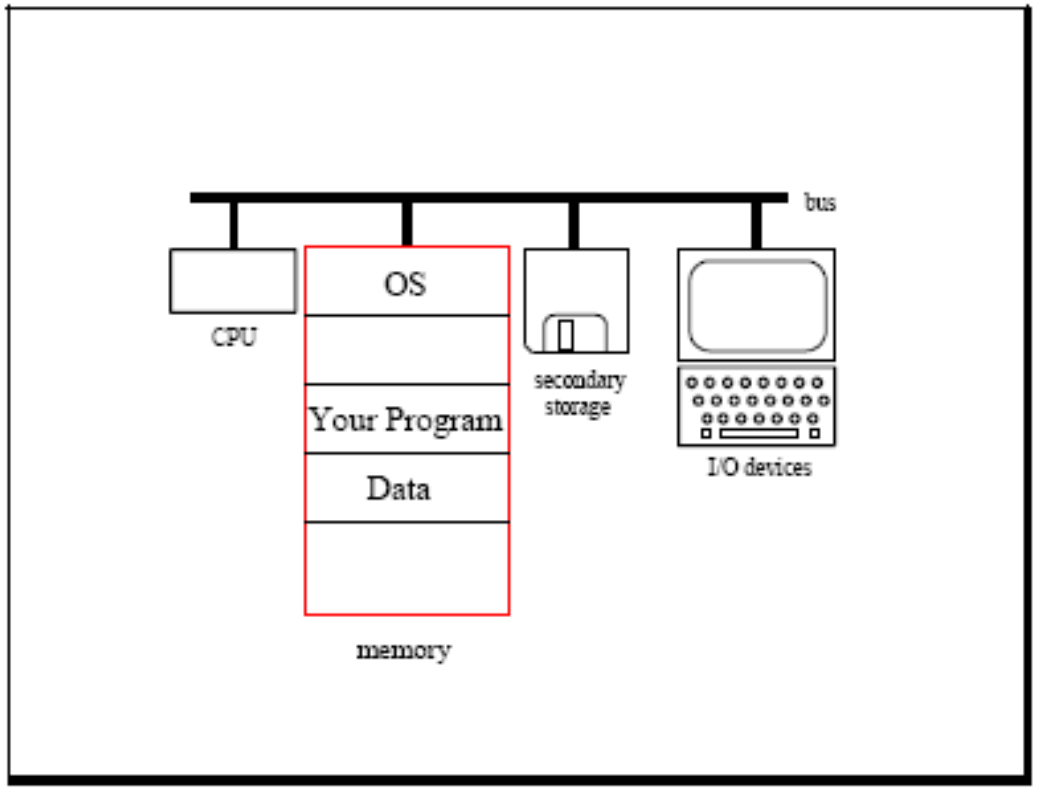
Lecture 6: 函数



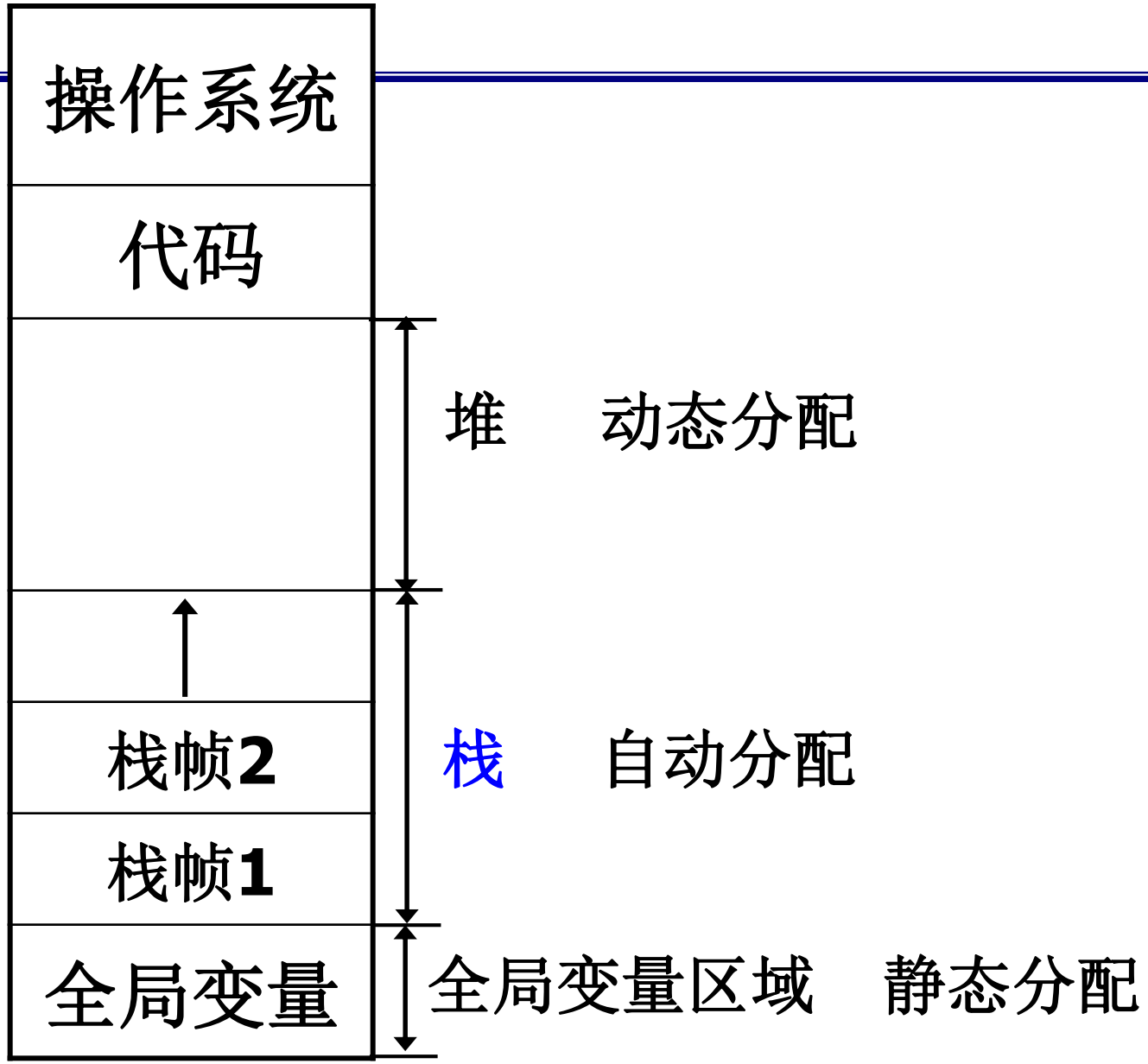
1. 函数概述
2. 函数的使用
3. **函数调用的实现过程**
4. 函数与设计

内存分布

- **存储程序原理，1945，John von Neumann**
- **把代码和数据都存放在内存中**



内存分布状况



```
#include <stdio.h>
int sum;
int Add(int a, int b);
int main()
{
    int x, y;
    scanf("%d %d", &x, &y);
    sum = Add(x, y);
    printf("%d", sum);
    return 0;
}
int Add(int a, int b)
{
    int result;
    result = a + b;
    return result;
}
```

```
sum:    0x004257B0

scanf: 0x00401160 ~ 0x004011BA
printf:0x004010E0 ~ 0x0040115B
Add:   0x004010A0 ~ 0x004010CA
main:  0x00401020 ~ 0x00401086

x:     0x0012FF7C
y:     0x0012FF78

a:     0x0012FF24
b:     0x0012FF28
result:0x0012FF18
```

函数调用的实现过程

控制流：程序当前执行位置的流向；

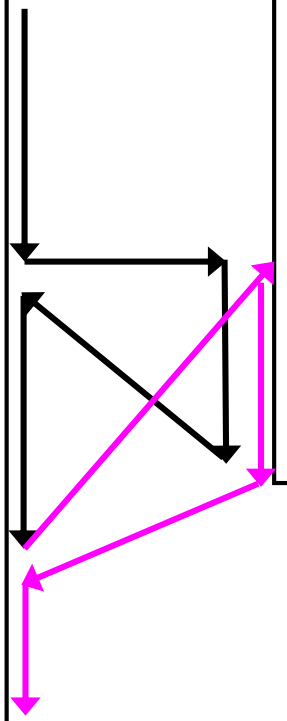
数据流：函数调用发生及结束时，数据在函数之间流转的过程。

当一个函数被调用时：

1. 在内存的栈空间当中为其分配一个栈帧，用来存放该函数的形参和局部变量；
2. 把实参的值复制到相应的形参变量；
3. 控制转移到该函数的起始位置；
4. 该函数开始执行；
5. 控制流和返回值返回到函数调用点，栈帧释放。

控制流的变化

```
int   main( )
{
   double   x,  y,  z;

   y  =  6.0;
   x  =  Area( y / 3.0 );

   ...

   z  =  3.4 * Area(7.88);

   ...
}
```



```
/* 给定半径，计算一个圆的面积 */
double Area(double r)
{
   return(3.14 * r * r);
}
```

一个简单的例子

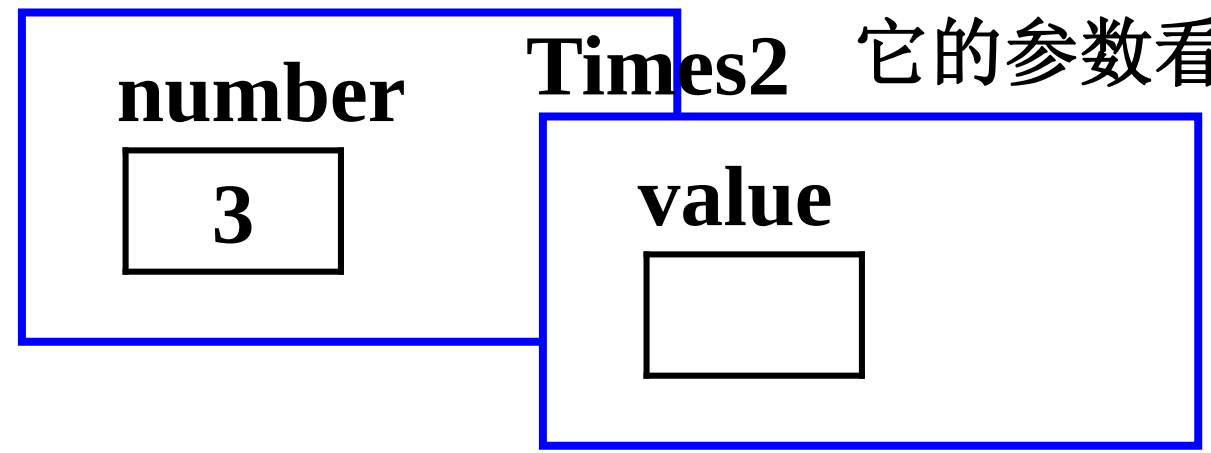
```
int  Times2(int  value);
int main ( )
{
        int  number;
        printf(“请输入一个整数：”);
        scanf(“%d”,  &number);
        printf(“该数的两倍是：%d”,  Times2(number));
}
int  Times2(int  value)
{
        return(2 * value);
}
```

main

number

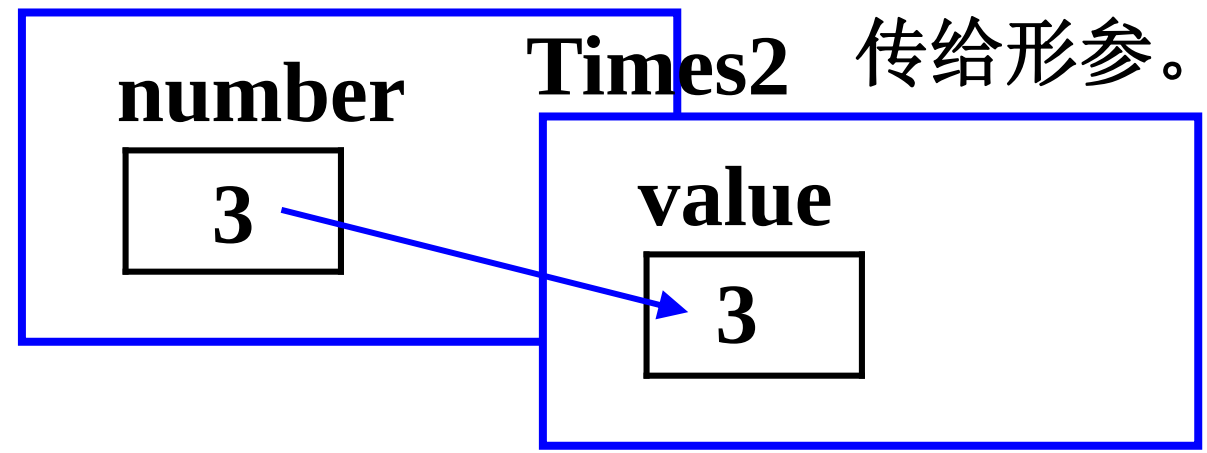
3

```
int  Times2(int  value);
int main ( )
{
        int  number;
        printf(“请输入一个整数：”);
        scanf(“%d”,  &number);
        printf(“该数的两倍是：%d”,  Times2(number));
}

int  Times2(int  value)
{
        return(2 * value);
}
```



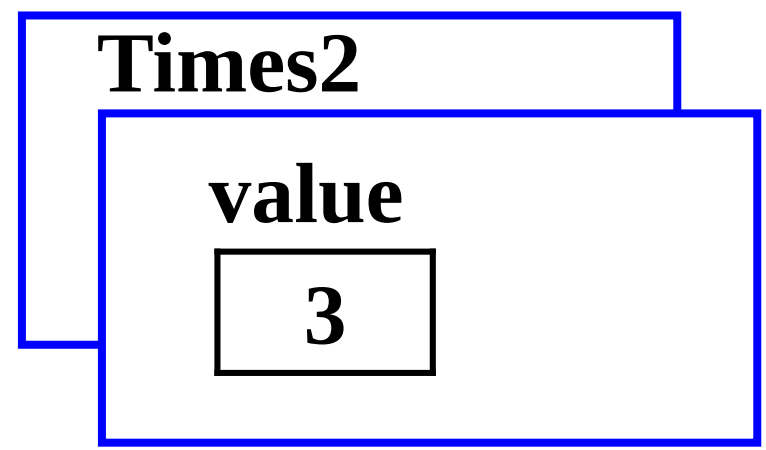
Times2也得到一个栈帧，它的参数看成局部变量

```
int  Times2(int  value);
int main ( )
{
        int  number;
        printf(“请输入一个整数：”);
        scanf(“%d”,  &number);
        printf(“该数的两倍是：%d”,  Times2(number));
}

int  Times2(int  value)
{
        return(2 * value);
}
```



“值传递”，把实参的值传给形参。

```
int  Times2(int  value);
int main ( )
{
        int  number;
        printf("请输入一个整数：");
        scanf("%d",  &number);
        printf("该数的两倍是：%d",  Times2(number));
}

int  Times2(int  value)
{
        return(2 * value);
}
```



把Times2的栈帧叠在主函数的栈帧之上，说明在执行Times2函数时，主函数中的变量是不可见的。

```
int  Times2(int  value);
int main ( )
{
        int  number;
        printf(“请输入一个整数：”);
        scanf(“%d”,  &number);
        printf(“该数的两倍是：%d”,  Times2(number));
}

int  Times2(int  value)
{
        return(2 * value);
}
```

6

main

number

3

Times2函数的返回值被放在函数的调用位置上，然后，分配给Times2函数的堆栈区域被释放。

示例1: Anagrams问题

问题描述：

Anagrams指的是具有如下特性的两个单词：在这两个单词当中，每一个英文字母（不区分大小写）所出现的次数都是相同的。例如，Unclear和Nuclear、Rimon和MinOR都是Anagrams。编写一个程序，输入两个单词（只包含英文字母），然后判断一下，这两个单词是否是Anagrams。每一个单词的长度不会超过80个字符，而且是大小写无关的。

编写(或调用)一个字符串大小写转换函数

字符串处理函数补充

（1）strlwr函数　　#include <string.h>

功能：将字符中的字母转化为小写 (lowercase)

格式：char * strlwr(char *str); //NOT const char *

说明：只转换str中出现的大写字母, 不改变其它字符。
返回指向str的指针。

例子：char str[ ] = “ABC";
strlwr(str); // “abc”

（2）strupr函数

功能：将字符中的字母转化为大写 (uppercase)

格式：char * strupr(char *str);

说明：只转换str中出现的小写字母, 不改变其它字符。
返回指向str的指针。

例子：char str[ ] = “abc";
strupr(str); // “ABC”

（3）tolower函数　　#include <ctype.h>

功能：将字母转化为小写形式

格式：int　tolower(int　c);

（4）toupper函数

功能：将字母转化为大写形式

格式：int　toupper (int　c);

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
char *strlwr2(char *s);
char *strupr2(char *s);
int main()
{
    char str[100] = "Copywrite 2013 Technologies";
    puts(str);

    strlwr(str);
    puts(str);

    strlwr2(str);
    puts(str);

    strupr2(str);
    puts(str);
    return 0;
}
```

```
char *strlwr2(char *s)
{
    if(s == NULL)  return NULL;
    char *p = s;
    while (*s != '\0') {
        *s = tolower(*s);
            s++;
    }
    return p;
}
```

Copywrite 2013 Technologies

copywrite 2013 technologies

copywrite 2013 technologies

COPYWRITE 2013 TECHNOLOGIES

```
char *strupr2(char *s)
{
    if(s == NULL)  return NULL;
    char *p = s;
    while (*s != '\0') {
        if((*s) >= 'a' && (*s) <= 'z')
            (*s) -= 32;  // ‘a’ – ‘A’ = 97 – 65 = 32
        s++;
    }
    return p;
}
```

2. 用指针作为形参可以改变实参内容

1. 为什么不能‘return s;’?

（5）itoa函数　　#include <stdlib.h>

功能：将整数value转化成radix进制表示的字符串str，进制radix在2~36之间，返回str指针

格式：char *itoa(int value, char *str, int radix);

（6）atoi函数

功能：将字符串str转化为整型数值

格式：int atoi(const char *str);

The function stops reading the input string at the first character that it cannot recognize as part of a number. [MSDN]

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    char buffer[65];
    int r;
    // An example of the itoa function.
    for( r=10; r>=2; r-- ) {
        itoa( -1, buffer, r );
        printf( "base %d: %s (%d chars)\n", r, buffer, strlen(buffer) );
    }
    char    *str = NULL;
    int     value = 0;
    // An example of the atoi function.
    str = "  -2309 pigs 12";
    value = atoi( str );
    printf( "Function: atoi( \"%s\" ) = %d\n", str, value );
    return 0;
}
```

base 10: -1 (2 chars)

base 9: 12068657453 (11 chars)

base 8: 37777777777 (11 chars)

base 7: 211301422353 (12 chars)

base 6: 1550104015503 (13 chars)

base 5: 32244002423140 (14 chars)

base 4: 3333333333333333 (16 chars)

base 3: 102002022201221111210 (21 chars)

base 2: 11111111111111111111111111111111 (32 chars)

Function: atoi(" -2309 pigs 12") = -2309