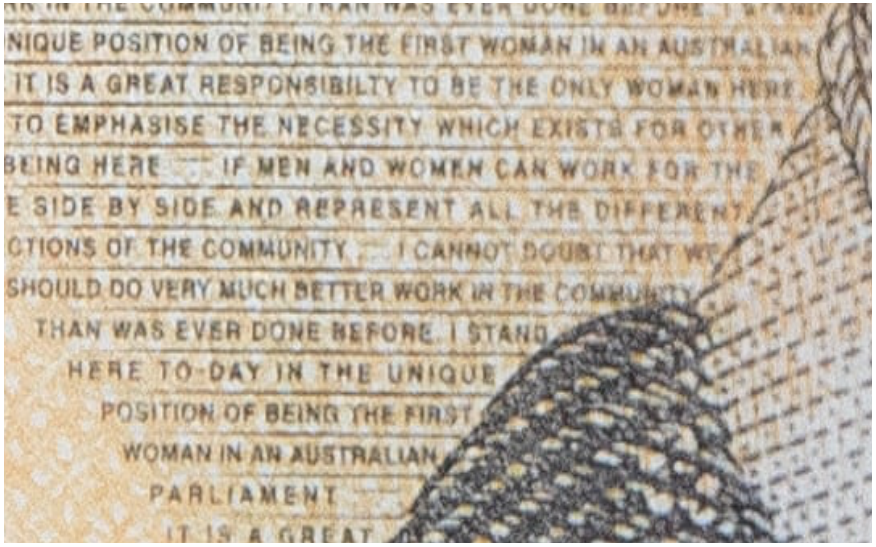
关于bug：肯定有，只是没有发现

- 澳洲去年 10 月发行的五十元新纸币被发现有错字
- 含有错字的纸币已经印制 4600 万张
- 纸币中的文字将 responsibility 印成了 responsibilty
- 澳大利亚央行发言人表示错字将在下一轮印刷中修正



School of Economics ang Management, Beihang University

Introduction to C Programming

Jichang Zhao

jichang@buaa.edu.cn

Pointers

Objectives

- Pointers
- 阅读《C语言程序设计现代方法》第2版第11章

How to swap two variables

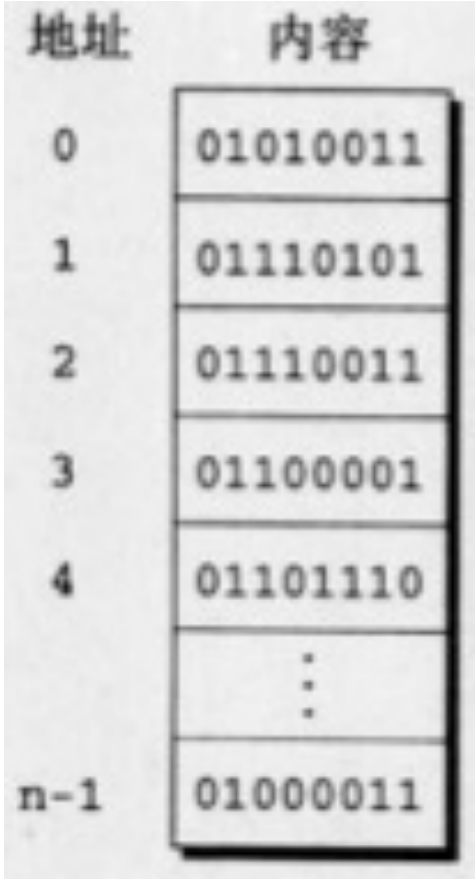
- `void swap(int a, int b)`
- `{`
  - `int temp=a;`
  - `a=b;`
  - `b=temp;`
- `}`
- Pass by value
  - A called function receives values from its calling function, stores the passed values in its own local parameters, manipulates these parameters appropriately, and directly returns, at most, a single value
  - 补充：传值的长、短处

Pass by Reference

- Passing an address
  - Called function can reference, or access, the variable using the passed address
  - Also referred to as a *call by reference* when the term applies only to those parameters whose addresses have been passed
  - 补充：这里跟C++的传引用不同，本质上还是传值（只不过这个值是一个地址），会产生地址的复本
  - 补充：C++里的写法是void swap(int&,int&);//传引用，无复本产生

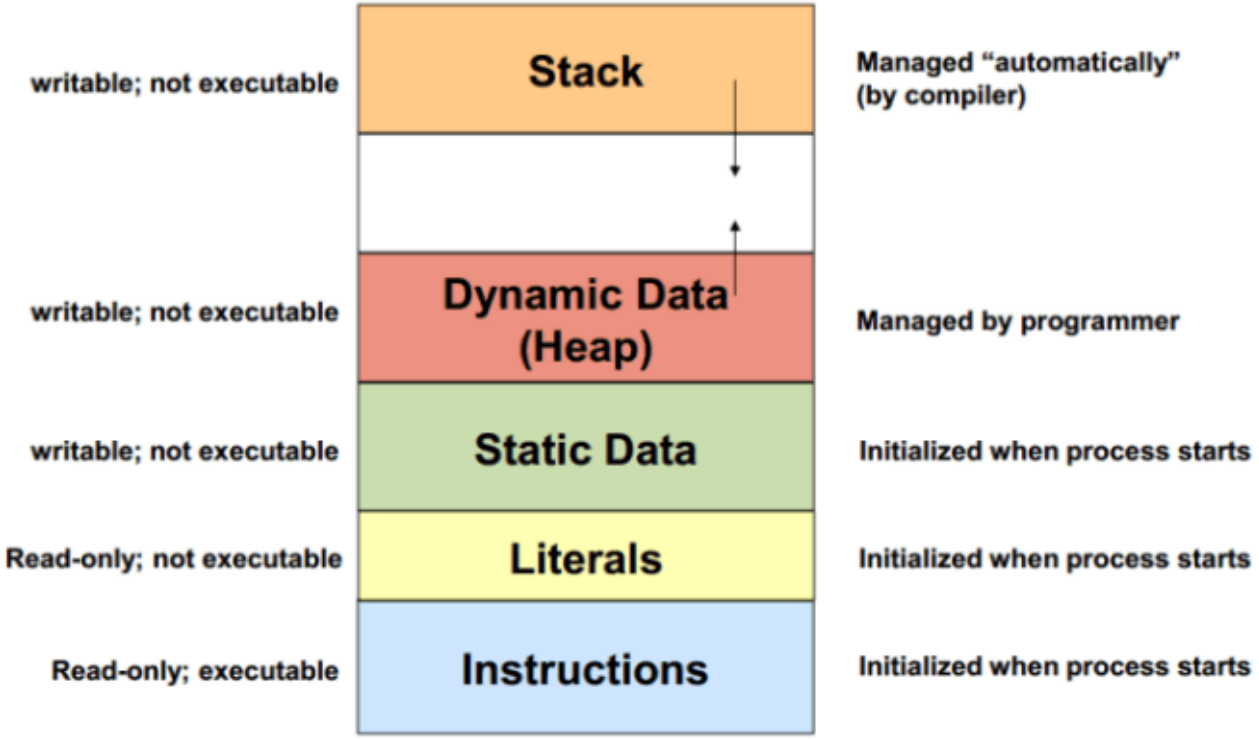
补充：内存的管理

- 通过编址进行访问
  - 编址，即内存的地址
  - 地址本质上是一个序号：可以看作是0~n-1的数
- 变量的地址
  - 存储变量的内存段的首字节序号
- 能不能用整数表示地址？
  - 范围可能不同：地址有上、下限，不一定“连续”
  - 地址有运算的约束
    - 加减的步长是类型相关的
    - 乘除不一定有意义
  - 地址的操作更敏感：会导致严重错误，系统视角
- 有必要用特殊的变量来表示地址
  - 指针变量（pointer variable)
  - 一般来讲，指针就是地址，指针变量存储（某种类型变量的）地址



补充：内存“图景”

- 注意栈（stack）和堆（heap）的区别
  - 都是内存，都可以通过地址来访问或操作



Using Addresses

- Indirection operator *
  - `*numAddr` means *the variable whose address is stored in* `numAddr`
  - Or, the *variable pointed to by* `numAddr`
- When using a pointer, the value obtained is always found by first going to the pointer for an address, which is called **indirect addressing**

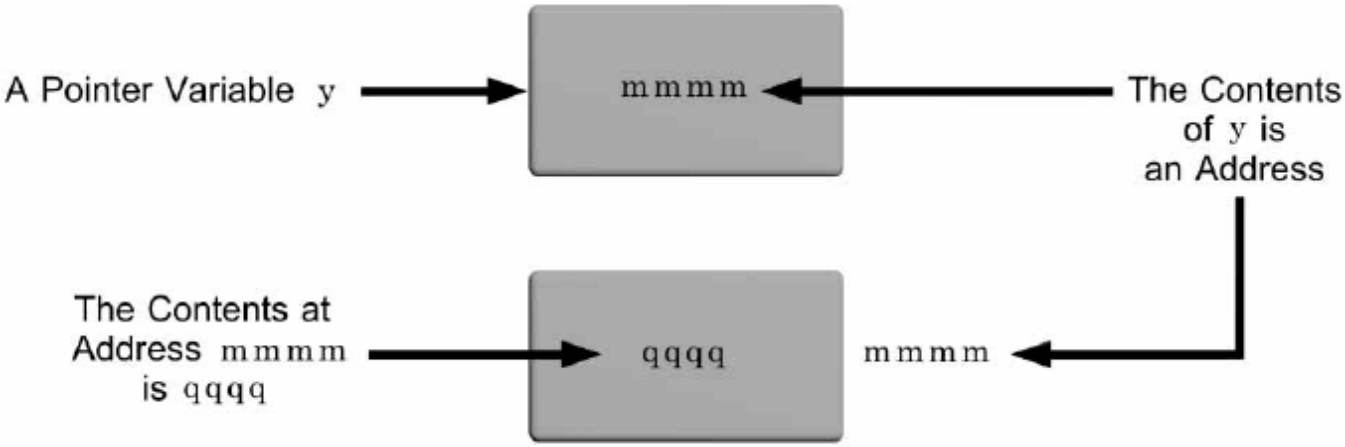


Figure 7.11 Using a pointer variable

Declaring and Using Pointers

- In declaring a pointer variable, C requires that we also specify the type of variable that is pointed to
  - `int *numAddr;`
  - 补充：为什么一定要约定指向变量的类型？（如图7.12）
  - 补充：C语言要求每个指针变量只能指向一种特定类型（引用类型）对象，但对引用类型本身没有限制（甚至可能是指针或函数）
- This declaration can be read in a number of ways
  - *the variable pointed to by `numAddr` is an integer*
  - *`numAddr` points to an integer*

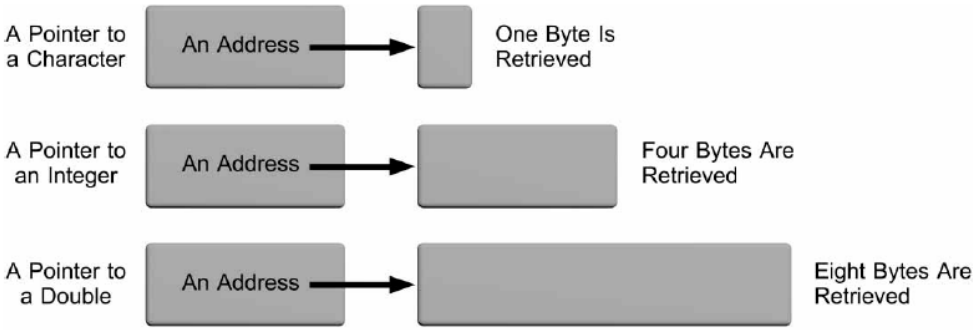


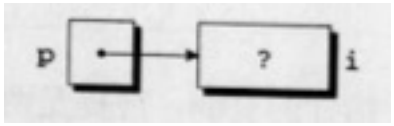
Figure 7.12 Addressing different data types using pointers

Pointer Initialization

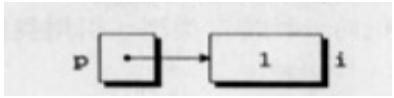
- Pointers can be initialized when they are declared:
  - `int miles;`
  - `int *ptNum = &miles;`
  - `double credits;`
  - `double *ptCre = &credits;`

补充：取址运算符与间接寻址运算符

- 取址运算符
  - &
  - 声明指针变量仅为指针分配空间，但并没有指向具体对象，可通过取址运算进行初始化
  - `int i, *p;`
  - `p=&i;`



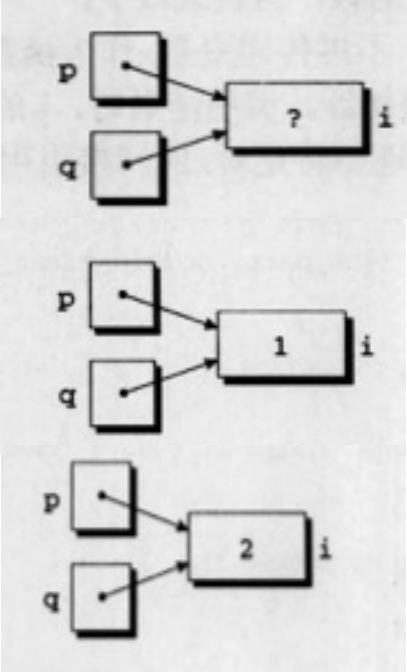
- 间接寻址运算符
  - *
  - 一旦指针变量指向了对象，就可以通过*访问对象中的内容
  - `*p=1;`
  - `printf("%d\n", *p);`



- 不要把间接寻址运算符用于未初始化的指针变量
  - 会导致未定义的行为（地址可能不合法）
  - 给*p赋值更加危险（万一地址合法，会有变量被莫名修改）

补充：指针赋值

- C语言允许使用赋值运算符进行指针的复制
  - 两个指针应具有相同的类型
  - `int i, j, *p, *q;`
  - `p=&i;`
  - `q=p;`
  - `*p=1;`
  - `*q=2;`
- 注意不要混淆
  - `*q=*p`与`q=p`不同



Manipulating Pointers

- A pointer, constructed either as a variable or function parameter, contains a value: an address
- By adding numbers to and subtracting numbers from pointers, we can obtain different addresses
- The addresses in pointers can be compared using any of the relational operators (==, !=, <, >, etc.)
- 补充：指针运算在数组和字符串中会继续讨论
- 补充：非法运算的示例(pa.c)

补充：指针作为参数

- 希望函数能够改变变量
  - 不再传递变量作为函数的实际参数，而提供变量的地址，即声明相应的指针形参
  - `void decompose(double x, long *int_part, double *frac_part)`
  - `{`
    - `*int_part=(long)x;`
    - `*frac_part=x-*int_part;`
  - `}`
  - `//Invoke`
  - `double x=3.14;//no change`
  - `long ipart;//will change`
  - `double fpart;//will change`
  - `decompose(x,&ipart,&fpart);`

补充：指针作为参数

- 对scanf函数的使用也可以发生变化
  - `char a;`
  - `char *p=&a;`
  - `scanf("%c",p);`
- 可能产生的常见错误
  - 对于指针参数传递实参时忘记&取址
- 指针变量本身还是是传值的
  - 如pv.c示例
  - 即指针变量所存储的地址不会被破坏（在地址复本上操作）
  - 但地址指向的值可能会被修改（所谓间接的本质）

补充：指针作为参数

- 指针作为参数通常是为了修改变量
  - 提供指针的主要目的
- 但有时是为了节省时间和空间
  - 如变量需要大量的存储空间，传递（复制）的成本很高
  - 可以只传一部分（如起始地址）
  - 这时不允许函数内部对指针指向的变量进行修改
- 使用const来表明函数不会改变指针参数所指向的对象
  - `void f(const int *p)`
  - 如果函数内部出现
  - `*p=0;`
  - 则编译器会发现
  - 注意：是阻止函数修改p指向的整数(*p)，并不阻止对p的修改
  - `int * const p;//用来保护p`

补充：指针作为返回值

- 函数可以返回指针
  - int *max(const int *a, const int *b)
  - {
    if(*a>*b)
    - return a;
  - else
    - return b;
  - }
  - //invoke
  - int *p, i,j;
  - i=10;
  - j=20;
  - p=max(&i,&j);

补充：指针作为返回值

- 需要注意
  - 函数可以返回指向全局变量的指针
  - 函数可以返回指向局部静态变量的指针
  - 函数不能返回指向自动局部变量的指针
    - 函数退出后自动变量生命周期结束
    - 指针将会无效
  - DEMO: pf.c

补充：指针可以指向指针

- 指针的指针
  - `int *p1,**p2;`
  - `int i;`
  - `p1=&i;`
  - `p2=&p1;`
  - 形成“指向”链
    - DEMO: pp.c

补充：空指针

- C里的空指针
  - NULL
  - #define NULL ((void *)0)，能够转换成指向任意类型
  - 是一种指针无效的约定
- 常用于“暂时”的初始化
  - 标记了初始状态为“空”(DEMO: pnull.c)
  - 在进行指针所指对象的访问前，可以通过判断是否为空来避免指向无效时导致的错误
  - `int i, *p=NULL;`
  - `p=&i;`
  - `if(p!=NULL)`
  - `{`
    - `*p=10;`
  - `}`

Homework

- (比较琐碎无聊，但请反复体会指针的使用）
- 1. P273 简答题 第2题
- 2. P275, 3
- 3. P275, 4
- 4. P275, 6
- 5. P281, 2
- 6. P281, 5
- 7. 编写并测试一个函数larger_of，该函数把两个double类型变量的值替换为较大的值 。如x>y时，y将被修改为x。