C++的基础知识

朱晓旭

苏州大学计算机科学与技术学院

计算机和二进制

- 二进制
  - 数字：125
  - 字符：’A’ “A”
  - 图片：bmp jpg
  - 声音：MP3
  - 视频：MP4 RMVB

原码，反码，补码

- 计算机中整数的表示:
- 原码：最高位为符号位，其余位数值部分，用二进制数的绝对值表示。

例如，+9的原码是00001001

└→符号位上的0表示正数

-9的原码是10001001。

└→符号位上的1表示负数

原码，反码，补码(续1)

- 反码：正数的反码与原码相同，负数的反码是保持符号位为1，然后将其原码的数值部分按位取反。
- 例如，-9的反码：因为是负数，则符号位为“1”；其余7位为-9的绝对值+9的原码0001001按位取反为1110110，所以-9的反码是11110110

原码，反码，补码（续2）

- 补码：
  - 正数的补码与原码相同
  - 负数的补码是将其原码除符号位外按位取反后加1得到
- 大多数计算机系统采用补码来表示整数
  - 使用补码，可以将符号位和其它位统一处理
  - 减法也可按加法来处理。

C++数据类型

- 数据类型
  - 数据以变量或常量的形式存在
  - 变量和常量都有数据类型
- 预定义数据类型
  - char、int、float、double
- 构造数据类型
  - 数组、指针、联合、结构、枚举、类等

C++数据类型

数据类型	关键字	字节数	数值范围
字　　符　　型	char	1	-128～127
整　　型	int	4	-2147483648～2147483647
单精度浮点型	float	4	±(3.4E-38～3.4E38)
双精度符点型	double	8	±(1.7E-308～1.7E308

C++数据类型

- **类型修饰符**
  - signed　　有符号
  - unsigned　　无符号
  - short　　短型
  - long　　长型

数据类型标识符	字节数	数值范围	常量写法举例
Char	1	-128 ~ 127	‘A’, ‘0’,’\n’
signed char	1	-128 ~ 127	56
unsigned char	1	0 ~ 255	100
short [int]	2	-32768 ~ 32767	100
signed short [int]	2	-32768 ~ 32767	-3456
unsigned short [int]	2	0 ~ 65535	0xff
int	4	-2147483648 ~ 2147483647	1000
signed int	4	-2147483648 ~ 2147483647	-123456
unsigned int	4	0 ~ 4294967295	0xffff

数据类型标识符	字节数	数值范围	常量写法举例
long [int]	4	-2147483648 ~ 2147483647	-123456
signed long [int]	4	-2147483648 ~ 2147483647	-3246
unsigned long [int]	4	0 ~ 4294967295	123456
float	4	±(3.4E-38 ~ 3.4E38)	2.35, -53.231 , 3E-2
Double	8	±(1.7E-308 ~ 1.7E308)	12.354,-2.5E10
long double	10	±(1.2E-4932 ~ 1.2E4932)	8.5E-300

变量定义和赋值

- 数据类型 变量名;
  - int number;
  - double mark;
- C++的变量必须先定义后使用
- 变量在定义时可以赋初值
  - int count=0;
  - 不赋初值会怎么样?

变量和输入输出

- cin>>
  - 把键盘的输入赋值给变量
- cout<<
  - 把变量的值输出到显示器

变量命名规则

- 规则
  - 允许字母、数字和下划线（_）三类符号
  - 且开头字符必须是字母或下划线
  - 变量名不能和C++中的关键字同名

- 注意：
  - C++中区分变量名的大小写
  - 变量名尽量做到“见名知意”

深度理解变量

- 变量
  - 变量名
  - 变量的值
  - 变量的地址

- 定义变量实质是分配内存（地址和大小）
- sizeof()计算变量所占的内存
- C和C++有一种特殊的变量
  - 指针变量

常量

- 整型常量
- 浮点型常量
- 字符型常量
  - ASCII
  - 字符的本质是数字
- 字符串常量

整数的溢出


```
#include <iostream>
using namespace std;

int main()
{
        short int num = 32766;
        cout << num << endl;
        num = num + 2;
        cout << num << endl;

    return 0;
}
```

男子交通卡现1700万天价消费

本报讯 （记者曹文清）上海又现公交卡“天价消费”。网友“猎兔星胖仔”近日在新民网上海滩微博上反映，自己在充值时发现公共交通卡内惊现一笔1700多万元的消费，让人匪夷所思。

“猎兔星胖仔”说，6月17日晚7时多，他在轨交8号线刷卡出站时发现卡里余额已为负数，便来到延吉中路站站厅的自动加值机前查询余额并充值，结果吓了一跳，“最近交易金额里有一串特别长的数字，仔细一数竟有1700多万元，实在太夸张了。”从网友拍下的图片上看，6月17日当天确实有一笔高达17448960元的交易。不仅如此，屏幕上的充值选项也不见了。一头雾水的他只能来到人工服务窗口，在充值30元现金后，交通卡又突然恢复正常。

事实上，“天价交通卡”在上海并非首例。2个月前，一市民在自助加值机上发现一笔高达1800万元的消费。3个月前，有网友称刷卡出站时，闸机显示卡内余额竟高达4000多万元。

为此，记者联系上海公共交通卡股份有限公司的工作人员。对方查询卡号后答复，其实网友的交通卡并未发生这笔巨额交易，而是这台自助充值机在显示过程中出现问题，不会影响卡片的正常使用。作为卡片管理方，他们已与自助加值机的管理方地铁公司联系，对机器进行维修。至于为何频频出现这类事情，对方解释说，可能是加值机系统存在差错，目前全市轨交站内的自助加值机正在逐步升级，系统更新后，类似差错会减少。

浮点数的舍入误差


```
#include <iostream>
using namespace std;

int main()
{
        double num1 = 11.000000000000000000001;
        double num2 = 11.010000000000000000001;

        cout << num1 << endl;
        cout << num2 << endl;

    return 0;
}
```

- 1990年2月25日，海湾战争期间，在沙特阿拉伯宰赫兰的爱国者导弹防御系统因浮点数舍入错误而失效，该系统的计算机精度仅有24位，存在0.0001%的计时误差，所以有效时间阙值是20个小时。当系统运行100个小时以后，已经积累了0.3422秒的误差。这个错误导致导弹系统不断地自我循环，而不能正确地瞄准目标。结果未能拦截一枚伊拉克飞毛腿导弹，飞毛腿导弹在军营中爆炸，造成28名美国陆军死亡。
- 1996年6月4日，在亚利安五号运载火箭发射后37秒，偏离预定轨道而炸毁。原因是软件系统试图将64位浮点数转换为16位浮点数，造成溢出错误。

运算符

- 算术运算
  - + - * / %
  - ++ --
  - -
- 赋值运算
  - =
  - 复合赋值运算符
- 关系运算
  - > < <= >= == !=
- 逻辑运算符
  - ! && ||

逻辑运算符的注意

- “短路”
  - 与短路
  - 或短路

条件运算符

- 唯一的三目运算符

逗号运算符

- 逗号运算符的运算优先级是最低的
- 一般形式为：
  - 表达式1，表达式2，……，表达式N
- 在计算逗号表达式的值时，按从左至右的顺序依次分别计算各个表达式的值，而整个逗号表达式的值和类型是由最右边的表达式决定

强制类型转换

- 将某一数据从一种数据类型向另一种数据类型进行转换。
- 其使用的一般形式：
  - 数据类型标识符 （表达式）
  - （数据类型标识符）表达式

```
int i=2;
float a,b;
a=float(i);
//将变量i的类型强制转换为浮点型，并将其值赋给变量a
b=(float)i;
//将变量i的类型强制转换为浮点型，并将其值赋给变量b
```

C++运算符的优先级和结合性

优先级	运算符	结合性
1	() : : [] -> . .* ->*	自左至右
2	! ~ ++ -- + - * & (类型) sizeof new[] delete[]	自右至左
3	* / %	自左至右
4	+ -	自左至右
5	<< >>	自左至右
6	< <= > >=	自左至右
7	== !=	自左至右
8	&	自左至右
9	^	自左至右
10	|	自左至右
11	&&	自左至右
12	||	自左至右
13	?:	自右至左
14	= += -= *= /= %= <<= >>= &= ^= |=	自右至左
15	,	自左至右

输入输出流

- #include <iostream>
- cout
  - cout<<数据1<<数据2<<……<<数据n;
- 说明:
- “<<”是流输出操作符，用于向cout输出流中插入数据
- cout的作用是向标准输出设备上输出数据，被输出的数据可以是常量、已有值的变量或是一个表达式

输入输出流（续）

- cin
- 其格式如下：
  - cin>>变量名1>>变量名2>>……>>变量名n;
- cin是系统预定义的一个标准输入设备
- cin的功能是：
  - 当程序在运行过程中执行到cin时，程序会暂停执行并等待用户从键盘输入相应数目的数据，用户输入完数据并回车后，cin从输入流中取得相应的数据并传送给其后的变量中
- “>>”操作符后除了变量名外不得有其他数字、字符串或字符，否则系统会报错

输出格式控制

- #include <iomanip>

控制符	功能
dec	十进制数输出
hex	十六进制数输出
oct	八进制数输出
setfill(c)	在给定的输出域宽度内填充字符
setprecision(n)	设显示小数精度为n位
setw(n)	设域宽为n个字符
setiosflags(ios::fixed)	固定的浮点显示

输出格式控制（续）

setiosflags(ios::scientific)	指数显示
setiosflags(ios::left)	左对齐
setiosflags(ios::right)	右对齐
setiosflags(ios::skipws)	忽略前导空白
setiosflags(ios::uppercase)	十六进制数大写输出
setiosflags(ios::lowercase)	十六进制数小写输出
setiosflags(ios::showbase)	按十六／八进制输出数据时，前面显示前导符0x／0；

三种基本结构

- 顺序
- 分支
- 循环

- 结构化程序设计

分支

- if
  - 三种if
  - 复合语句（花括号）
- Switch
  - 多路分支
  - break

循环

- 三种循环
  - for
    - 适合已知次数
  - while
    - 适合未知次数，至少0次
  - do-while
    - 适合未知次数，至少1次
- 注意for循环的终值

流程控制语句

- continue;
- break;
- goto;

- return

循环嵌套

- 三种循环可以互相嵌套
- 9X9乘法表

- 穷举法
  - 提高效率

数组

- 一组相同类型的多个变量
- 数组的本质是一段连续内存
- C和C++数组不做越界检查

函数

- 独立设计
- 代码重用

- 函数设计
  - 函数的参数
  - 函数的返回值

- 孪生素数

内联函数

- 声明时使用关键字 inline。
- 编译时在调用处用函数体进行替换,节省了参数传递、控制转移等开销。
- 注意：
  - 内联函数体内不能有循环语句和switch语句。
  - 内联函数的声明必须出现在内联函数第一次被调用之前。
- 是请求，不是命令
- 以空间换时间

内联函数例子


```
#include<iostream.h>
inline double CalArea(double radius)
{
  return 3.14*radius*radius;
}
int main ( )
{
  double r(3.0);
  double area;
  area=CalArea(r);
  cout<<area<<endl;
  return 0;
}
```

具有缺省参数值的函数

- 函数在声明时可以预先给出默认的形参值
  - 调用时如给出实参，则采用实参值
  - 否则采用预先给出的默认形参值

```
void delay(int x=1000)
{   for(int i=0;i<x;i++)
    ;
}
```

```
void main(void)
{
    delay(5000);
    delay();
}
```

具有缺省参数值的函数（续1）

- 可以提供多个参数的，但必须靠右原则
  - 否则导致二义性
- 例：

```
int add(int x,int y=5,int z=6) //正确
int add(int x=1,int y=5,int z) //错误
int add(int x=1,int y,int z=6) //错误
```

具有缺省参数值的函数（续2）

- 也可以通过函数声明的方式提供缺省参数值

```
int add(int x=5,int y=6);
void main(void)
{
    add（）; //调用在定义前
}
int add(int x,int y)
{   return x+y;   }
```

指针

- 指向其余变量地址的变量
- 间接带来灵活
- 指针和数组
- 有了指针可以动态申请内存
  - new
  - delete
- 指针变量所占用的内存和指针类型无关

引用

- 引用是别名

```
int i,j;
int &ri=i;
        //建立一个int型的引用ri,并将其
        //初始化为变量i的一个别名
j=10;
ri=j;//相当于 i=j;
```

- &与取地址符类似
  - 如何区分?
- 引用的类型和被引用类型应该相同

引用（续）

- 声明一个引用时，必须同时对它进行初始化，使它指向一个已存在的对象。
- 一旦一个引用被初始化后，就不能改为指向其它对象。
- C语言函数参数的传递
  - 按值传递：最多
  - 按地址传递：靠数组和指针实现
- C++添加了引用传递参数

例 输入两个整数交换后输出


```
#include<iostream.h>
void Swap(int& a, int& b);
int main ( )
{
 int x(5), y(10);
 cout<<"x="<<x<<"     y="<<y<<endl;
 Swap(x,y);
 cout<<"x="<<x<<"     y="<<y<<endl;
 return 0;
}
```

Swap用传值方法编写


```
void Swap(int a, int b)
{
 int t;
 t=a;
 a=b;
 b=t;
}
```

运行结果(不能交换)

x=5　y=10

x=5　y=10

Swap用传址方法编写

```
void Swap(int* a, int* b)
{
  int t;
  t=*a;
  *a=*b;
  *b=t;
}
```

Main中调用语句需要修改为

Swap(&x,&y)

运行结果:

x=5 y=10

x=10 y=5

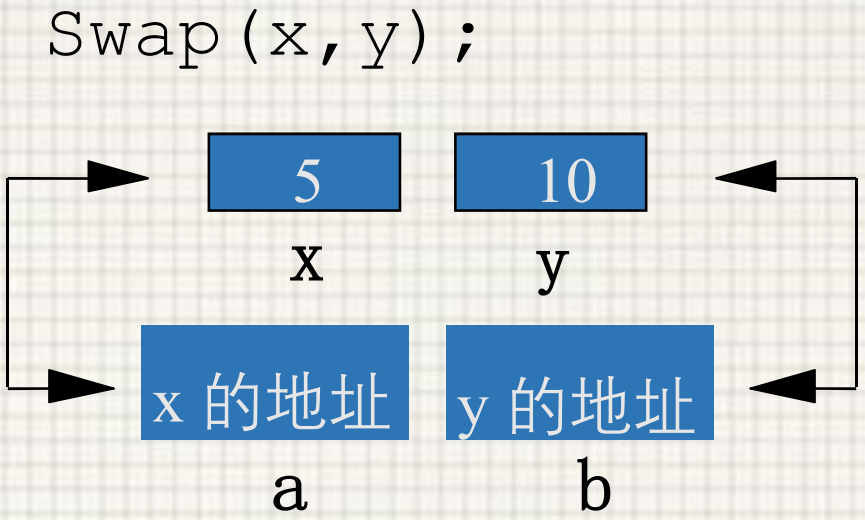
Swap用传引用方法

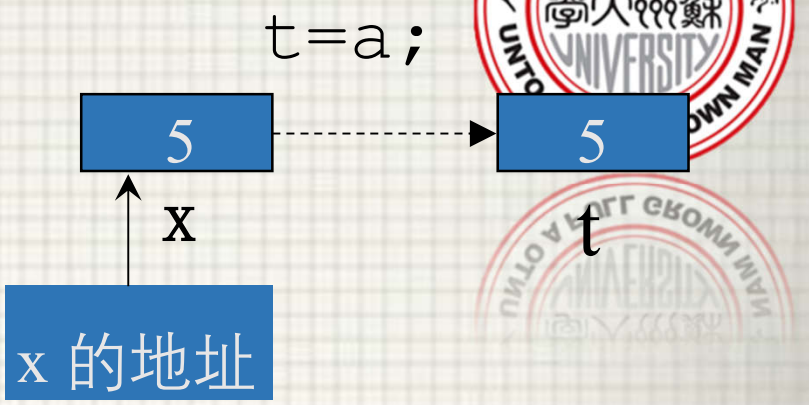
```
void Swap(int& a, int& b)
{
 int t;
 t=a;
 a=b;
 b=t;
}
```

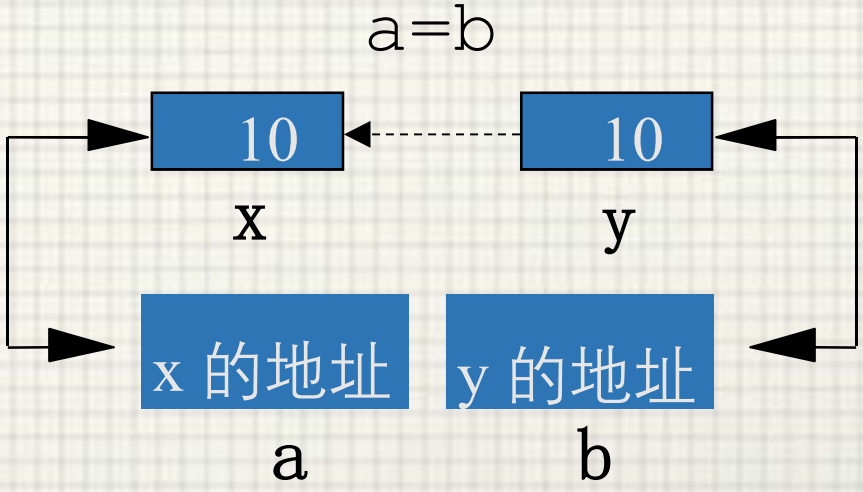
运行结果:

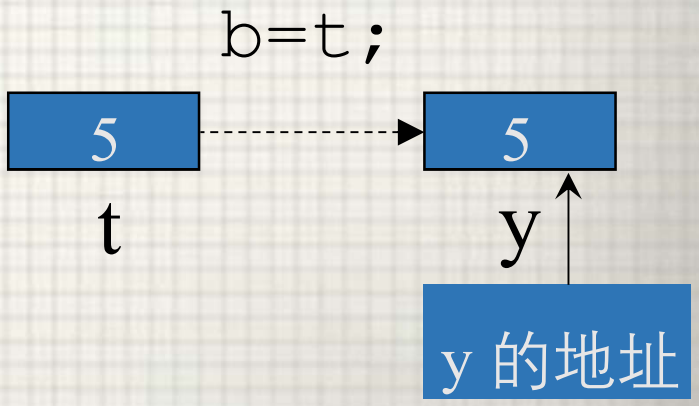
x=5 y=10

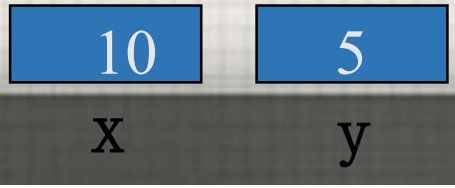
x=10 y=5


Swap(x,y);
5
10
x
y
x 的地址
y 的地址
a
b


t=a;
5
5
x
t
x 的地址
a


a=b
10
10
x
y
x 的地址
y 的地址
a
b


b=t;
5
5
t
y
y 的地址
b


10
5
x
y

引用（续）

- 引用作为形式参数的优点
  - 书写简单
  - 高效（节约存储）
  - 可以从函数中带回值
    - 函数需要返回多个值，可以使用

注释

- C语言注释
  - /*　*/
- C++注释
  - /*　*/　　注释一段
  - //　　注释一行或一行的后半部分
- 更加灵活
- 良好注释可以增强程序可读性
- 注释是调试程序的常用手段

重载

- 面向对象的重要特征之一
- 静态的多态
- 所谓重载就是同一个符号在不同的上下文中代表不同的含义。
- 分类
  - 重载函数
  - 重载运算符
- 方便使用，便于记忆

一个实际问题

- C语言中求绝对值
  - int abs(int)
  - double fabs(double)
  - long labs(long)
  - 函数的调用者负担大
- 能否三个函数具有相同名称?
  - 重载函数

重载函数

- **功能相近的函数在相同的作用域内具有相同函数名**
- 要求：
  - 函数形式参数类型不同
    - int add(int x, int y);
    - float add(float x, float y);
  - 函数形式参数个数不同
    - int add(int x, int y);
    - int add(int x, int y, int z);

绝对值函数的重载实现


```
int abs(int x)
{
   return x>0?x:-x;
}
double abs(double x)
{
   return x>0?x:-x;
}
long abs(long x)
{
   return x>0?x:-x;
}
```

绝对值重载函数的调用


```
#include <iostream.h>
void main()
{
  int x1=32;
  double x2=-156.4;
  long x3=-45768L;
  cout<<abs(x1)<<endl;
  cout<<abs(x2)<<endl;
  cout<<abs(x3)<<endl;
}
```

重载函数注意事项

- **不可以通过函数形式参数的名称进行重载**
  - int add(int x, int y);
  - int add(int a, int b);
- **不可以通过函数返回值的类型进行重载**
  - int add(int x, int y);
  - long add(int x, int y);
- 不要将不同功能的函数声明为重载函数

误用重载的例子

```
int add(int x,int y)
{
    return x+y;
}

float add(float x,float y)
{
    return x-y;
 }
```

重载典型例题


```
void Display(char * string) {……}
void Display(long value) {……}
void Display(double value) {……}
void main()
{
  Display("\nPrintthis,please!");
  Display(123456789);
  Display(3.14159);
}
```

函数模板

- 创建通用函数的途径
  - 因为很多算法的处理过程本质上与数据类型无关
- 可以大大降低代码冗余
- 函数模板定义形式
  - Template <类型参数表>
  - 返回值类型　函数名称（形参列表）
  - {
    - 函数体
  - }

函数模板典型例题

- 求绝对值问题
  - 前面用了三个重载函数
  - int abs(int x)
  - double abs(double x)
  - double abs(double x)
- 用模板技术解决求绝对值问题

```
template <typename T>
T abs(T x)
{
        return x>0?x:-x;
}
```

模板与重载比较

- 模板
  - 功能相近或类似的多个函数
  - 函数体中代码类似（只是数据类型有区别）
  - 大大减少了代码重复
  - 例如前面的求绝对值函数
- 重载
  - 功能相近或类似的多个函数
  - 函数体中代码不同
  - 例如前面的Display问题

模板与重载总结

- 没有重载函数，调用复杂
- 没有模板，程序会变得冗长和呆板
- 重载是为函数调用者服务的
- 模板是为函数编写者服务的
- 二者各有所长

动态分配内存

- C
  - malloc free
  - 函数
- C++
  - new delete
  - 运算符
- 提供了一种完成动态数组功能的方法

const指针

- 指向常量的指针
  - const <类型> * <指针变量>
  - <类型> const * <指针变量>
  - 指针所指向的对象不可以修改
- 常量指针
  - <类型> * const <指针变量>
  - 指针本身不能修改，但其指向的对象可以修改
- 指向常量的常量指针
  - <类型> const * const <指针变量>
  - const <类型> * const <指针变量>
  - 指针与所指东西全部为常量

向量

- vector
- 使用方式
  - #include <vector>
  - vector <typename> 变量名
- C++的安全数组
  - 能防止越界
  - 自己知道自己的大小
  - 能动态改变大小
- 具体成员函数，大家自己查阅

字符串string

- 可以用等号直接赋值
- 可以直接判相等
- 可以知道自己的长度
- 自己直接连接字符串
- 相关成员函数大家自行查阅