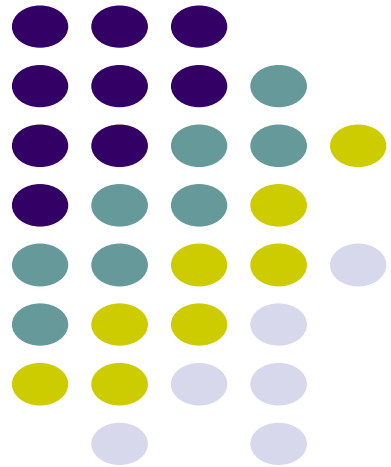
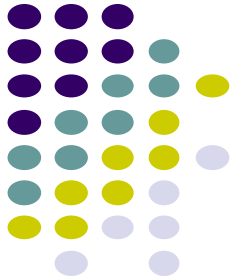
《计算机系统基础（四）：编程与调试实践》

从程序员角度认识系统



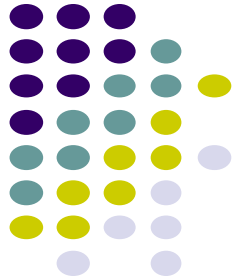
计算机系统基础—从程序员角度认识系统



- 目标：

  培养学生的系统能力，使其成为一个“高效”程序员，在程序调试、性能提升、程序移植和健壮性等方面成为高手；建立扎实的计算机系统概念，为后续的OS、编译、体系结构等课程打下坚实基础。

实例1——字面量比较



ISO C90标准下，在32位系统上 以下C表达式的结果是什么？

-2147483648 < 2147483647

false（与事实不符）！

ISO C99标准下为true

以下关系表达式结果呢？

int i = -2147483648;

i < 2147483647

true！

理解该问题需要知道：

编译器如何处理字面量

高级语言中运算规则

高级语言与指令之间的对应

机器指令的执行过程

机器级数据的表示和运算

……

-2147483647-1 < 2147483647，结果怎样？

编译器处理常量时默认的类型

- C90

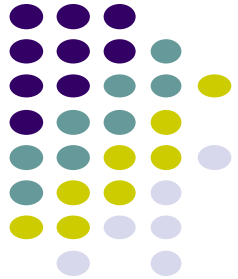
范围	类型
$0 \sim 2^{31}-1$	int
$2^{31} \sim 2^{32}-1$	unsigned int
$2^{32} \sim 2^{63}-1$	long long
$2^{63} \sim 2^{64}-1$	unsigned long long

2^{31}=2147483648，机器数为：100…0（31个0）

- **C99**

范围	类型
$0 \sim 2^{31}-1$	int
$2^{31} \sim 2^{63}-1$	long long
$2^{63} \sim 2^{64}-1$	unsigned long long

实例1——字面量比较



ISO C90标准下，在32位系统上 以下C表达式的结果是什么？

-2147483648 < 2147483647

false（与事实不符）！

ISO C99标准下为true

以下关系表达式结果呢？

int i = -2147483648;

i < 2147483647

true！

理解该问题需要知道：

编译器如何处理字面量

高级语言中运算规则

高级语言与指令之间的对应

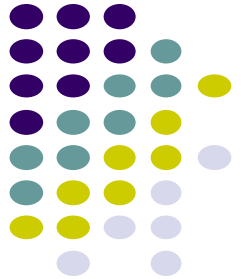
机器指令的执行过程

机器级数据的表示和运算

……

-2147483647-1 < 2147483647，结果怎样？

实例2——运算规则



当用len=0调用sum函数时，其返回值应该是多少？

```
sum(int a[ ], unsigned len)
{
    int  i，  sum = 0;
    for (i = 0; i <= len–1; i++)
          sum += a[i];
    return sum;
}
```

当参数len为0时，返回值应该是0，但是在机器上执行时，却发生访存异常。但当len为int型时则正常。为什么?

理解该问题需要知道：

高级语言中运算规则

机器指令的含义和执行

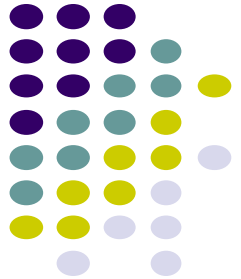
计算机内部的运算电路

异常的检测和处理

虚拟地址空间

……

实例3——整数乘法运算及溢出



```
/* 复制数组到堆中，count为数组元素个数 */
int copy_array(int *array, int count) {
        int i;
       /* 在堆区申请一块内存 */
        int *myarray = (int *) malloc(count*sizeof(int));
       if (myarray == NULL)
              return -1;
       for (i = 0; i < count; i++)
              myarray[i] = array[i];
       return count;
}
```

当count=2^{30}+1时， 程序会发生什么情况？

理解该问题需要知道：

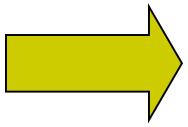
乘法运算及溢出

虚拟地址空间

存储空间映射

……

当参数count很大时，则count*sizeof(int)会溢出。如count=2^{30}+1时，count*sizeof(int)=4。



堆（heap）中大量数据被破坏！

实例4——强弱符号

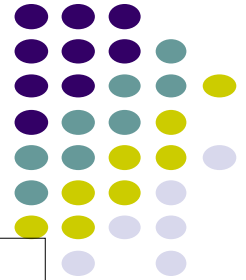
main.c

```
int d=100;
int x=200;
int main()
{
    p1( );
    printf ( “d=%d, x=%d\n” , d, x );
    return 0;
}
```

p1.c

```
double d;

void p1( )
{
    d=1.0;
}
```



打印结果是什么？

d=0，x=1 072 693 248

Why？

理解该问题需要知道：

机器级数据的表示

变量的存储空间分配

数据的大端/小端存储方式

链接器的符号解析规则

……

实例4——强弱符号

打印结果：d=0，x=1 072 693 248

double型数1.0对应的机器数3FF0 0000 0000 0000H

IA-32是小端方式

	0	1	2	3
&x	00	00	F0	3F
&d	00	00	00	00

高

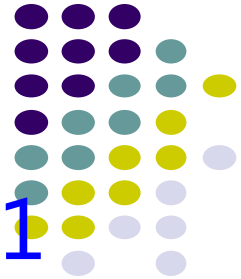
低

$2^{30}-1-(2^{20}-1)=2^{30}-2^{20}$

=1024*1024*1023

=1 072 693 248

实例5——浮点运算指令



- 使用老版本gcc –O2编译时，程序一输出0，程序二输出却是1

程序一：

```
#include<stdio.h>
double f(int x){
        return 1.0 / x;
}
void main(){
        double a,b;
        int i;
        a=f(10);
        b=f(10);
        i=a==b;
        printf("%d\n",i);
}
```

程序二：

```
#include<stdio.h>
double f(int x){
    return 1.0 / x;
}
void main(){
    double a,b,c;
    int i;
    a=f(10);
    b=f(10);
    c=f(10);
    i=a==b;
    printf("%d\n",i);
}
```

实例5——浮点运算指令

```
double f(int x)
{
    return 1.0 / x ;
}
```

```
8048328:    55          push   %ebp
8048329:    89 e5       mov    %esp,%ebp
804832b:    d9 e8       fld1
804832d:    da 75 08    fidivl 0x8(%ebp)
8048330:    c9          leave
8048331:    c3          ret
```

两条重要指令的功能如下。

fld1：将常数1压入栈顶ST(0)

fidivl：将指定存储单元操作数M[R[ebp]+8]中的int型数转换为double型，
再将ST(0)除以该数，并将结果存入ST(0)中

f(10)=0.1

0.1=0.00011[0011]B= 0.00011 0011 0011 0011 0011 0011 0011…B

实例5——浮点运算指令

```
08048334 <main>:
 8048334:      55                    push   %ebp
 8048335:      89 e5                 mov    %esp,%ebp
 8048337:      83 ec 08              sub    $0x8,%esp
 804833a:      83 e4 f0              and    $0xfffffff0,%esp
 804833d:      83 ec 0c              sub    $0xc,%esp
 8048340:      6a 0a                 push   $0xa
 8048342:      e8 e1 ff ff ff        call   8048328 <f> //计算a=f(10)
 8048347:      dd 5d f8              fstpl  0xfffffff8(%ebp) //a存入内存
 804834a:      c7 04 24 0a 00 00 00    movl   $0xa,(%esp)
 8048351:      e8 d2 ff ff ff        call   8048328 <f> //计算b=f(10)
 8048356:      dd 45 f8              fldl   0xfffffff8(%ebp) //a入栈顶
 8048359:      58                    pop    %eax
 804835a:      da e9                 fucompp      //比较ST(0)a和ST(1)b
 804835c:      df e0                 fnstsw %ax  //把FPU状态字送到AX
 804835e:      80 e4 45              and    $0x45,%ah
 8048361:      80 fc 40              cmp    $0x40,%ah
 8048364:      0f 94 c0              sete   %al
 8048367:      5a                    pop    %edx
 8048368:      0f b6 c0              movzbl %al,%eax
 804836b:      50                    push   %eax
 804836c:      68 d8 83 04 08        push   $0x80483d8
 8048371:      e8 f2 fe ff ff        call   8048268 <_init+0x38>
 8048376:      c9                    leave
 8048377:      c3                    ret
```

```
…
a = f(10) ;
b = f(10) ;
i = a == b;
…
```

80位→64位

64位→80位

0.1是无限循环小数，无法精确表示，因而，比较时，a舍入过而b没有舍入过，故 a≠b

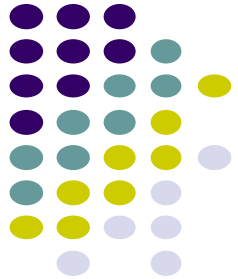
实例5——浮点运算指令

```
8048342:    e8 e1 ff ff ff     call   8048328 <f> //计算a
8048347:    dd 5d f8           fstpl  0xfffffff8(%ebp) //把a存回内存
                                      //a产生精度损失
804834a:    c7 04 24 0a 00 00 00    movl   $0xa,(%esp,1)
8048351:    e8 d2 ff ff ff     call   8048328 <f> //计算b
8048356:    dd 5d f0           fstpl  0xfffffff0(%ebp) //把b存回内存
                                     //b产生精度损失
8048359:    c7 04 24 0a 00 00 00    movl   $0xa,(%esp,1)
8048360:    e8 c3 ff ff ff     call   8048328 <f> //计算c
8048365:    dd d8              fstp   %st(0)
8048367:    dd 45 f8           fldl   0xfffffff8(%ebp) //从内存中载入a
804836a:    dd 45 f0           fldl   0xfffffff0(%ebp) //从内存中载入b
804836d:    d9 c9              fxch   %st(1)
804836f:    58                 pop    %eax
8048370:    da e9              fucompp //比较a , b
8048372:    df e0              fnstsw %ax
```

```
…
a = f(10) ;
b = f(10) ;
c = f(10) ;
i = a == b;
…
```

0.1是无限循环小数，无法精确表示，因而，比较时，a和b都是舍入过的，故 a=b！

小结



- 本次课介绍了几个C语言实例，并说明了用系统思维来分析编程问题的重要性。当然，要学会用系统思维方式来分析问题还需要我们掌握扎实的基本概念以及熟练使用各种工具！