CS0449

Lecture 1 [Introduction]

Why C? { 汇编/机器代码的高级抽象 => 全面控制cpu/右硬件
内存的低级抽象 => 全面控制内存

Lecture 2 [Data Representation]

Binary Encoding: $d_n d_{n-1} \cdots d_1 d_0 : d_n \times B^n + d_{n-1} \times B^{n-1} + \cdots + d_1 \times B^1 + d_0 \times B^0$

(B决定几进制, 所能表示最大数为 $B^n - 1$)

bit: 最小的二进制数字, 单位为小写b

byte: 8-bit value, 单位为大写B

nibble: 半字节, 4-bit, 41=二进制数

word: 对CPU而言最合适的尺寸, "32-bit CPU" (word size)

另: windows, x86的word为16-bit

计算机将电信号转成0或1

- UTF-16用于Java字符串
- 最通用是ASCII, 8-bit number表示字母

十六进制: F ⇔ 15 ⇔ 1111

二进制与十六进制的转换: 4个2进制 symbol 转 1个16进制 symbol

数字有限, 0000意味无法表示, 太大了

⊗ 二进制负数的转换

例. $-3 = -(3) \iff 3 = -(-3)$

$3_{10} = 0011_2$ | $-3_{10} = 1101_2$

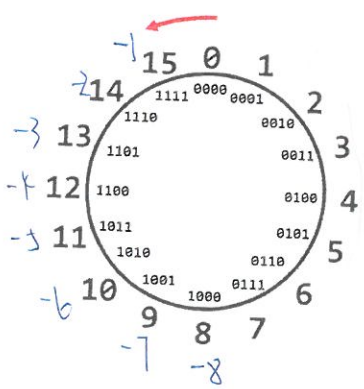
flip $= 1100_2$ | flip $= 0010_2$

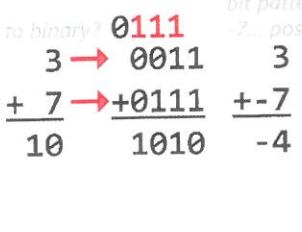
$-3 = \text{flip} + 1 = 1101_2$ | $3 = \text{flip} + 1 = 0011_2$

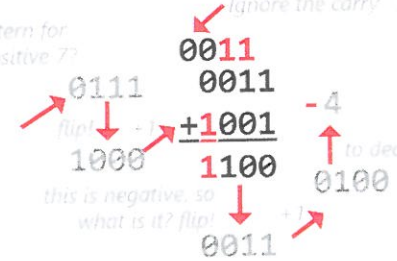
二进制加法直接相加, 类似...

二进制数 = -(其逆位 + 1)

先取原码, 再得反码, 最后补码





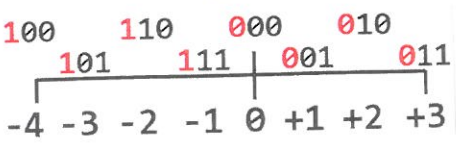


11010100 → -(00101011+1) → -(43+1) = -44

00100110 = 00100110 = 38

00000000 = 00000000 = 0

11111111 = -00000000 = -(0+1) = -1



(负数补1, 正数补0)

关于正负二进制数取原码

正数补零 00100110 = 38

0000000000100110 = 38

负数补一 10100110 = -90

1111111110100110 = -90

-0000000001011001 = -90

10100110 = -90

-(01011001+1) = -90

-01011010 = -90

sign extension

top bit(MSB) 决定正负

Java 获取负数值的方法

```java
public class AbsTest {
  public static int abs(int x) {
    if (x < 0) {
      x = -x;
    }

    return x;   // 取绝对值
  }
  public static void main(String[] args) {
    System.out.println(
      String.format("|%d| = %d", Integer.MIN_VALUE, AbsTest.abs(Integer.MIN_VALUE))
    );
  }
}
                                   // Outputs: |-2147483648| = -2147483648
```

最小 integer

C中的Integer

C允许将变量声明为有符号或无符号。

无符号整数的范围是 $0 - 2^n - 1$

(有符号 $2^{n-1} - 1$)

由变量尺寸决定

大多情况我们用 int

Integer类型 (signed by default)

- char 8 bits (byte)
- short int 16 bits
- int 32 bits (word)
- long int 64 bits

```c
#include <limits.h>   // 提供 INT_MAX 等..
#include <stdio.h>    // 提供 printf

int main() {
  printf("%d ", INT_MAX);    // 打印最大的signed int
  printf("%u\n", UINT_MAX);  // 打印最大的unsigned int
  return 0;
}                  // Output:
```

=> 2147483647 4294967295

C允许有符号整数变量与无符号整数变量的转换: casting

Fractional Binary 分数二进制

$(0110.1101)_2$

$2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

0×8 + 1×0.5 +
1×4 + 1×0.25 +
1×2 + 0×0.125 +
0×1 + 1×0.0625 +

$= 6.8125_{10}$

6.8125_{10}

$6 \div 2_{10} = 3R0$

$3 \div 2_{10} = 1R1$

$0.8125_{10} \times 2 = 1.6250$

$0.6250_{10} \times 2 = 1.2500$

$0.2500_{10} \times 2 = 0.5000$

$0.5000_{10} \times 2 = 1.0000$ final.

110.1101

小数部分化二进制:

×2取整,直到1

0.1_{10}如何转换二进制？（会无尽循环）

位操作 sum

x|y	or
x&y	and
x^y	xor
!x	not
~x	补-
x<<y	逻辑左移
x>>y	逻辑右移

Lecture3 [Data Representation II]

Bit Manipulation 位操作

Bitwise operations 将值视为 patten of bits

1) 逻辑否定 NOT

A	Q
0	1
1	0

结果为 ¬A，$\overline{A}$；C语言中的操作符为"!"

2) 逻辑乘积 AND

A&B A∧B AB

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

（只有都是1时才1，只有A B都满足…）

and，在C中为&

3) 逻辑或 OR

A|B，A∨B

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

or，在C中为|

（或，只要有一个是1，结果为1）

4) 逻辑差异（两者须不同，结果为1）

C语言为^

A^B，A⊕B

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

5) Bit Shifting：左移，空位补0

B=A<<4（把A左移4 bits，得到B）

同样可以right shift

正补0，负补1，未位是.

左移补0相当于×2. shifting left by n = multiplying by 2^n.

a<<n等于 $a*2^n$

右移相于/2. shifting right by n = dividing by 2^n.

a>>n等于 $a/2^n$

Fractional Encoding (接lec2)

fixed-point representation 定点表示 (小数点固定且隐藏).

小数点四处移动: floating point (浮点计数法?).

Floating-Point Number Representation 浮点数表示

基数为科学计数法: 例. -3.9×10^{-3} → exponent

sign, significand

IEEE754: 二进制浮点数算术标准 → 特征形式: $(+/-)1.f \times 2^{exp}$ → 指数

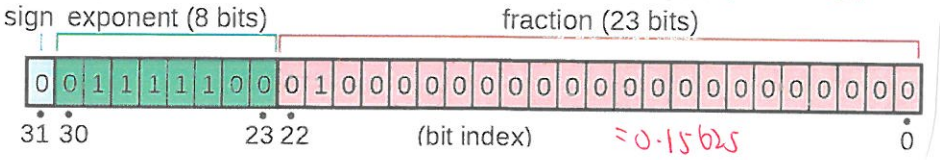
$-0.001\ 010 = -1.010 \times 2^{-3}$

(二进制小数点前为1). 有效数

32-bit float format

存储隐式实为24bits.

fraction越多, 精度越高
exponent越多, 范围越广



fraction只存储二进制小数点后的数字
二进制小数点前的1是隐式的(规范化表示)

exponent field使用 biased notation

biased representation = exponent + bias constant

Biased = 127 + exp (-126 ~ +127) 以下单精度浮点数.

↓ (1~254).

$(-1)^S \times 1.f \times 2^{exp-127}$

例. $+1.001\ 0101 \times 2^{+7}$

sign = 0 正数.

Biased exp = exp + 127 = 134
= 10000110

fraction = 00101011

s	E	f
0	10000110	00101010000000000000000

$(-1)^0 \times 1.001\ 0101 \times 2^{134-127}$

另例. -1.010×2^{-3}

sign = 1 负.

Biased exp = -3 + 127 = 124 = 01111100.

fraction = 010

s	E	f
1	01111100	01000000000000000000000

$(-1)^1 \times 1.010 \times 2^{124-127}$

/ 将整数编为浮点数

以-2471为例,

化二进制 0000 1001 1010 0111$_2$

① 将其转换科学计数法

$1.001\ 1010\ 0111 \times 2^{+11}$

② Biased exponent = 11 + 127 = 138 = 1000 1010$_2$

③ fraction移入

s	exponent	fraction
0	10001010	00110100111000000000000

↑ 正数位 ↑ fraction移入.

举例：-12.59375_{10}

s	exponent	fraction
1	10000010	10010011000000000...000

负数

1）计算 biased exponent：$127+3=130$

$=10000010_2$

1）转换二进制：整数部分 1100_2

小数部分 0.10011_2

$\Downarrow (1100.10011)_2$

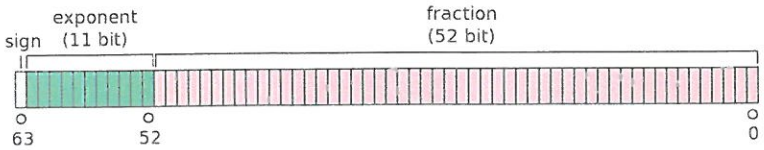
$\Downarrow 1.100100011 \times 2^3$

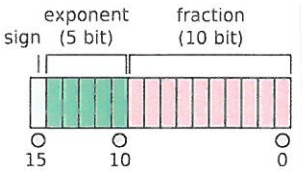
2）科学计数法：↑

```
public class FloatTest {
  public static void main(String[] args) {
    double var = Double.MAX_VALUE;

    while (var == Double.MAX_VALUE) {

      var = var + 0.1;
    }

    System.out.println("This never prints.");
  }
}
```

// 无法执行，超出限制

其它常见的形式：双精度 & 半精度

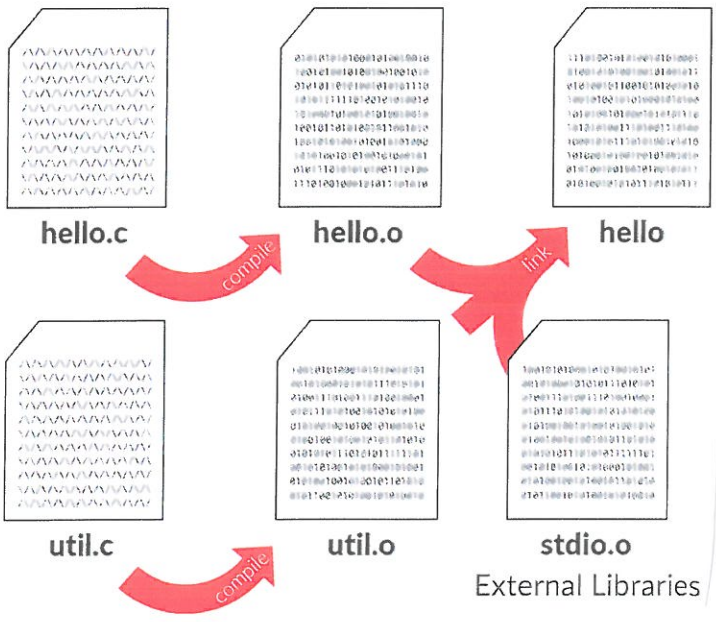




Lecture 4 [Introduction to C]

C是汇编语言。（C语言转化为机器码，Java先转化成字节码再机器码，而Python只是解释）。

C编译器将.c源文件转成.o对象文件；然后将其组合在一起形成可执行文件



编译器把.c转成机器代码.o

对象文件(.o)不可执行 = (object file)

util.c，util.o包含一些公共功能

（有时需要调用）

最终，每个部分合在一起形成可执行文件。

※ Linking

C与Python各有不足之处。

C是编译性语言而Python是解释性语言

- (complier + Linker)将代码转成机器码；
- 而机器码可以被OS加载，直接被硬件运行。

（新版硬件需要再编译）。

- 解释性语言由其它语言(C)而编写，
- 不依赖硬件，便捷。

而Java将源文件译成byte-code，需要虚拟机模拟这种架构。

	C(99)	vs Java
语言类型	面向功能	面向对象
编程单元	Function	Class
编译	gcc hello.c 创建机器代码	javac Hello.java 创建虚拟机语言bytecode
执行	a.out 输出并执行程序	java Hello. 解释bytecode

hello world输出

```c
#include<stdio.h>
int main(void) {
   printf("Hello World\n");
   return 0;
}
```

```java
public class HelloWorld {
   public static void main(String[] args) {
   System.out.println("Hello World");
   }
}
```

	C(99)	Java
存储	手动内存分配	自动 (garbage collection)
注释	//	//
常量	#define ,const	final
预处理器	✓	X
变量声明	在块的开始	使用之前
变量命名规则	sum_of_squares	sumOfSquares
访问库	#include <stdio.h>	import java.io.File;

```c
//包括printf函数的声明
#include <stdio.h>

//首先要执行的代码的主要功能
int main(void) {
  //打印字符串的方法；\n意为换行
  printf("Hello World\n");

  //返回0意味成功
  return 0;
}
```

2) "主"方法.

```c
#include <stdio.h>  //文件包含在文件顶部

int main(void) {  //你的代码首先要执行的主方法.
                  当没有实参时调用void.
  return 0;  //return 0成功；return an int (a word)来报错.
}
```

C的语法.

1) C的预处理器/不应当对待成机器代码，而是当别的.

```c
#include "hello.h"   //将本地文件转储到此位置
#include <stdio.h>   //将系统路径文件转储

#define DEBUG 0  //简单的文本替换

#if ( DEBUG )  //条件下编译特定代码
     …
#else
     …
#endif
```

3) 声明变量

```c
int main(void) {  //变量通常在函数顶部声明
  int n = 5;  //使用"="初始化
  return 0;   没有初始化，变量值是任意的.
}
```

4) Casting 转换

```c
int main(void) {
  int n = -50000;
  char smaller = n;
  unsigned int just_nonsense = (unsigned int)n;
  return 0;
}
```

//初始化时,给定的文本被强制为该类型.

//可以强制变量间的值,不管是否有意义.

//显式转换

显式转换就是强制转换
在被转换的表达式前加float,如(float)5,就把5转换为float类型.

5) 整数大小,复习:sizeof

```c
#include <stdio.h>
#include <stddef.h>

int main(void) {

  size_t int_byte_size = sizeof(int);
  size_t uint_byte_size = sizeof(unsigned int);

  printf("sizeof(int): %lu\n", int_byte_size);
  printf("sizeof(unsigned int): %lu\n", uint_byte_size);

  return 0;
}
```

//提供"printf"函数

//提供"size_t"类型.

//'sizeof'从宏观上给了byte的size

//'size_t'由C语言提供,用于计算大小.

output: sizeof(int): 4
sizeof(unsigned int): 4

(单位是byte).

Output:

sizeof(x):	(bytes)
char:	1
short:	2
int:	4
unsigned int:	4
long:	8
float:	4
double:	8

```c
#include <stdio.h>   // Gives us 'printf'

int main(void) {
  printf("sizeof(x):     (bytes)\n");
  printf("char:          %lu\n", sizeof(char));
  printf("short:         %lu\n", sizeof(short));
  printf("int:           %lu\n", sizeof(int));
  printf("unsigned int:  %lu\n", sizeof(unsigned int));
  printf("long:          %lu\n", sizeof(long));
  printf("float:         %lu\n", sizeof(float));
  printf("double:        %lu\n", sizeof(double));
  return 0;
}
```

各语言中整数尺寸的对比(大多32 bits)

sizeof(long long) ≥ sizeof(long) ≥ sizeof(int) ≥ sizeof(short)

一是中,short>=16 bits, long >= 32 bits,以上任何一个都可以为64 bits

6) Constants 常量

```c
const float PI = 3.1415;

int main(void) {

  float angle = PI * 2.0;

  PI = 3.0;

  return 0;
}
```

//可以用常量代替文字

X 隐式转换不行.

7) 枚举.

enum { }.

```c
#include <stdio.h>

enum { CS445, CS447, CS449 };

int main(void) {

  int my_class = CS449;

  printf("%d\n", my_class);

  return 0;
}
```

//位置分配从0而起

//print 2. CS449位于位置2

Output - 2

8)操作符

```c
int main(void) {
  int a = 5, b = -3, result;
  //'='初始化布署
  ,
  result = a + b + (a - b); //加,减
  result = a * b / (a % b); //乘除.模运算
  result = a & b | ~(a ^ b); //and, or, 仅. xor
  result = a << b;          //left shift
  result = a >> b;          // right shift .

  return 0;
}
```

此增强赋值操作符

```c
int main(void) {
  int a = 5, b = -3;

  a += b;   //相当于 a = a+b
  a *= b;   //相当于 a = a*b
  a &= b;   //相当于 a = a&b
  a <<= b;  //相当于 a = a<<b
  a++;      //递增, a = a+1
  a--;      //递减 a = a-1 .

  return 0;
}
```

9)隐式转换

```c
char a = 0x76;
short b = 0x5610;
c = (a & b)
printf("%lu\n", sizeof(a & b));
```

Output: 4 //1byte, 转换出int

C会隐式的先进行转换

(转成最适合的类型)

at least int .

C语法,控制流

流量控制: BS型算法.

```c
int main(void) {
  int a = 5, b = -3;

  if (a >= 5) {      //传统的布尔表达式
    printf("A\n");
  }
  else //单条语句不需要{}.
    printf("B\n");
    printf("Always happens!\n")

  return 0;
}
```

C不提供 Boolean type. (但有 <stdbool.h>

其实是个int型

```c
int a = 5, b = -3, result;

result = a <= b; //false时为结果为0; true时不为0
result = a > b;
result = a == b;
result = a != b;
```

多用int型, 为0则结果是危险的.

```c
if (0) {    // Always false
  printf("Never happens.\n");
}

if (-64) {  // Always true
  printf("Always happens.\n");
}
```

Loop循环. 大多与Java相似

for循环的一般模式.

for (initialization; loop invariant; update statement)

例. for (int i=0; i<10; i++)

循环体的一些特殊语句能改变流

continue 将结束当前迭代并开始下一个迭代

break 将完全退出循环.

while, do-while, for

```c
int main(void) {
  int i = 0;
  while (i < 10) {
    i++;
  }

  i = 0;
  do {
    i++;
  } while (i < 10);

  for (i = 0; i < 10; i++) {
  }

  return 0;
}
```

条件后置

一句

正确放置 break 指能让 switch 语句正确工作

一个个 case 匹配表达式开始，直至遇见 break.

若没有 break，将"fall through"其它 case 语句

```c
switch (character) {
  case '+': ...   // Fall through其它case
  case '-': ... break;
  case '*': ... break;
  default: ... break;
}
```

// 当不匹配之任何case

SUM [一个statement为一个block{}]

满足expression，执行statement

1) 条件块. if. else.

- if (expression) *statement*
- if (expression) *statement*
  else *statement*

另. if句可以包括接.

```
if (expression) statement
else if (expression) statement
else statement
```

2) 常见循环

- while (expression) *statement*

另.

- do
  statement
  while (expression);

3) For循环

- for (statement; expression; statement)
- continue; // 跳化到循环主体末尾
- break; // 无论条件如何，立即退出.

4) Switch

- switch (expression) {
  case const1: *statements*
  case const2: *statements*
  default: *statements*
  }
- break;

```c
int number_of_people(void) {
  return 3;
}

void news(void) {
  printf("no news");
}

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

// 声明返回名称前，声明返回类型.

? 当没有东西可返回时使用 void

? void也用于明确表示没有参数

void应当明确.

// 函数必须在使用前声明.

定义聚合数据类型.

struct 可以混合多种数据类型.

```c
struct Song {
  int lengthInSeconds;
  int yearRecorded;
};

struct Song my_song;
my_song.lengthInSeconds = 512;
```

// 定义2个int型变量

// 句点号

// 定义了一个Song类变量.

创建一个 Song 类型，用关键词 typedef 定义新类。

```
typedef struct {
  int lengthInSeconds;
  int yearRecorded;
} Song;

Song my_song;
my_song.lengthInSeconds = 512;
```

// Song 在之后写。

typedef 如何用无穷，也可以用来定义 integer 类型的变量 bool

```
typedef int bool;
```

// typedef 也可用于 enum 枚举类型。

```
typedef enum { CS 445, CS 447, CS449} Course;
```

（可以接受不同的值，枚举值，数值...）

```
void print_course(Course course) {
  switch (course) {
    case CS449: printf("The best course: CS449!\n");
  }
}
```

C 中变量记得初始化。

C 中没有 class

```
#include <stdio.h>
#include <stdlib.h>

void undefined_local() {
  int x;
  printf("x = %d\n", x);
}

void some_calc(int a) {
  a = a % 2 ? rand() : -a;
}

int main(void) {
  for (int i = 0; i < 5; i++) {
    some_calc(i * i);
    undefined_local();
  }
  return 0;
}
```

// stdlib.h 提供 rand 函数，会生成返回随机 int 值。

主方法，这边打印了5次。

Output:

x = 0
x = 1804289383
x = -4
x = 846930886
x = -16

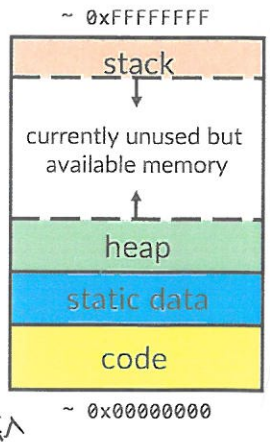
Lecture 5 [Introduction to Memory]

C 的内存模型

- 内存是一系列连续的 bits，能在逻辑上分为 byte（字节）或 word（字）。
- 将其视为 byte-addressable —（允许读取单个字节）

对于 byte-addressable memory，每个 byte（8 bits）都有自己的独特的 address。

（从 0 开始，address 具有数据增加而逐步上升）。



程序主要有两个部分：data & code。

```
int my_static_var = 1;

int factorial(int n) {
  if (n <= 1) { return my_static_var; }
  return n * factorial(n - 1);
}

void main(void) {
  factorial(5);
}
```

> 栈分配允许递归

红框 code 一般不易改变，在程序开始前载入

蓝框 static dat. 会改变，也在程序开始前载入。

[但是 data 与 section 的大小是固定的]

Stack 栈是临时动态数据的空间。

- 调用函数时分配，返回时释放
- 保存局部变量与函数实参