211: Computer Architecture
Spring 2017

Instructor: Prof. Santosh Nagarakatte

Topics:

- C programming conclusion
- Data Representation
  - Reading: Chapter 2.1, 2.2, 2.3, 2.4, 2.5

File I/O

A file is a contiguous set of bytes

- Has a name
- Can create, remove, read, write, and append

Unix/Linux supports persistent files stored on disk

- Access using system calls: open(), read(), write(), close(), creat(), lseek()
- Provide random access
- Section 2 of online manual (man)

C supports extended interface to UNIX files

- fopen(), fscanf(), fprintf(), fgetc(), fputc(), fclose()
- View files as streams of bytes
- Section 3 of online manual (man)

fopen

The fopen (pronounced "eff-open") function associates a physical file with a stream.

```
FILE *fopen(char* name, char* mode);
```

First argument: `name`

- The name of the physical file, or how to locate it on the storage device. This may be dependent on the underlying operating system.

Second argument: `mode`

- How the file will be used:
  `"r"` -- read from the file
  `"w"` -- write, starting at the beginning of the file
  `"a"` -- write, starting at the end of the file (append)

fprintf and fscanf

Once a file is opened, it can be read or written using fscanf() and fprintf()

These are just like scanf() and printf() except with an additional argument specifying a file pointer

- fprintf(outfile, "The answer is %d\n", x);
- fscanf(infile, "%s %d/%d/%d %lf",
  &name, &bMonth, &bDay, &bYear, &gpa);

When started, each executing program has three standard streams open for input, output, and errors

- stdin, stdout, stderr

Summary of C Programming

C is a language close to the hardware

- Ideal for building system software

- We looked at the basic aspects of C
  - Control flow – loops and break/continue statements
  - Pointers, structures, arrays
  - Memory management – malloc and free
  - File I/O operations

iClicker Pop Quiz 1

```
scanf(“%d”, val);
```

A: Reads an integer

B: Reads an long integer

C: Reads a floating point number

D: Segmentation fault

iClicker Pop Quiz 2

```
int *matvec(int **A, int *x) {
  int *y = malloc(N*sizeof(int));
  int i, j;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      y[i] += A[i][j]*x[j];
  return y;
}
```

A: The program computes Y = Ax and is correct

B: The program intends to compute Y = Ax but is wrong

C: The program experiences segmentation fault

D: The program is completely wrong

iClicker Pop Quiz 3

```
int **p;
p = (int**) malloc(N*sizeof(int));
for (i=0; i<N; i++) {
   p[i] = (int*) malloc(M*sizeof(int));
}
```

A: Wrong because it accesses uninitailized pointer

B: Wrong because it does not allocate correct space

C: Correct and initializes a 2-D matrix

D: Correct with no errros

iClicker Pop Quiz 4

```
int **p;
p = (int**) malloc(N*sizeof(int *));
for (i=0; i<=N; i++) {
  p[i] = (int *) malloc(M*sizeof(int));
}
```

A: Correct and reads a 2-D matrix

B: Buffer overflow

C: Memory can potentially leak

D: Segmentation fault

iClicker Pop Quiz 5

```
int *search(int *q, int val) {
  char* p = q;
  while (*p && *p != val)
     p += sizeof(int);
  return (int *) p;
}
```

A eger in the array

B: Correct and searches for an integer every until it encounters a 0.

C: Looks at every 4th integer and searches until it encounters a 0.

D: Buffer overflow always

iClicker Pop Quiz 6

```
int *foo () {
   int val;
   return &val;
}
```

A: Wrong because of a dangling pointer

B: Returns the address of the location in function foo

C: Buffer overflow

D: No errors

iClicker Pop Quiz 7

```
x = (int*) malloc(N*sizeof(int));
…
free(x);

y = (int *) malloc(M*sizeof(int));
…
free(x);
```

A: Correct

B: Double free

C: Buffer overflow

D: segmentation fault

iClicker Pop Quiz 8

```
x = (int*) malloc(N*sizeof(int));
…
y = x;
free(y);
…
```

A: Correct

B: Double free

C: Buffer overflow

D: segmentation fault

iClicker Pop Quiz 9

```
x = (int*) malloc(N*sizeof(int));

…

y = x+4;

free(y);

…
```

A: Correct

B: Double free

C: Buffer overflow

D: Segmentation fault

iClicker Quiz 10

```
x = (int*) malloc(N*sizeof(int));
<manipulate x>
free(x);
...
y = (int*) malloc(M*sizeof(int));
for (i=0; i<M; i++)
y[i] = x[i]++;
```

A: Double free

B: use-after-free error on y

C: Buffer overflow

D: use-after-free error on x

iClicker Quiz 11

```
foo() {
  int *x = (int *) malloc(N*sizeof(int));
  ...
  return;
}
```

A: Correct

B: Program hangs due to memory leaks

C: Program hangs in a long running program due to memory leaks

D: Segmentation fault

iClicker Pop Quiz 12

```
struct list { int val; struct list *next;};
void foo() {
  struct list *head = (struct list *) malloc(sizeof(struct list));
  head->val = 0;
  head->next = NULL;
  <create and manipulate the rest of the list>
  ...
  free(head);
return;
}
```

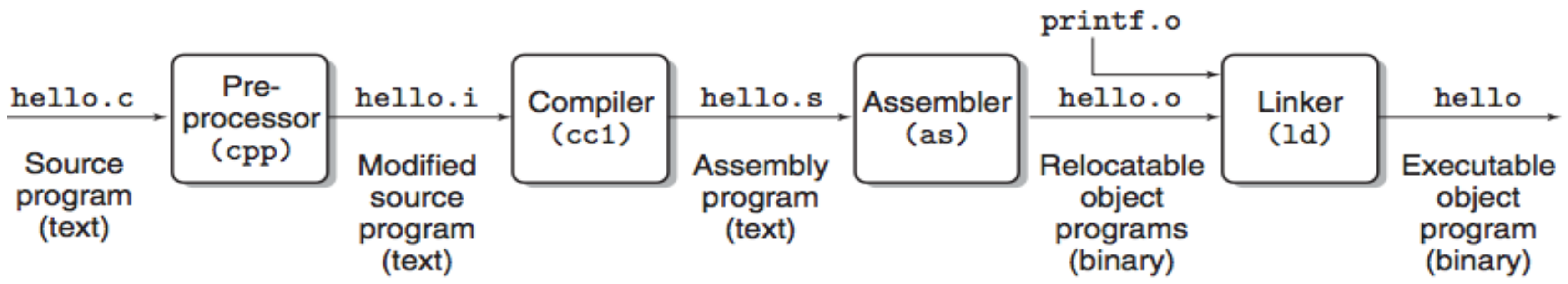
A: Correct

B: Memory leak

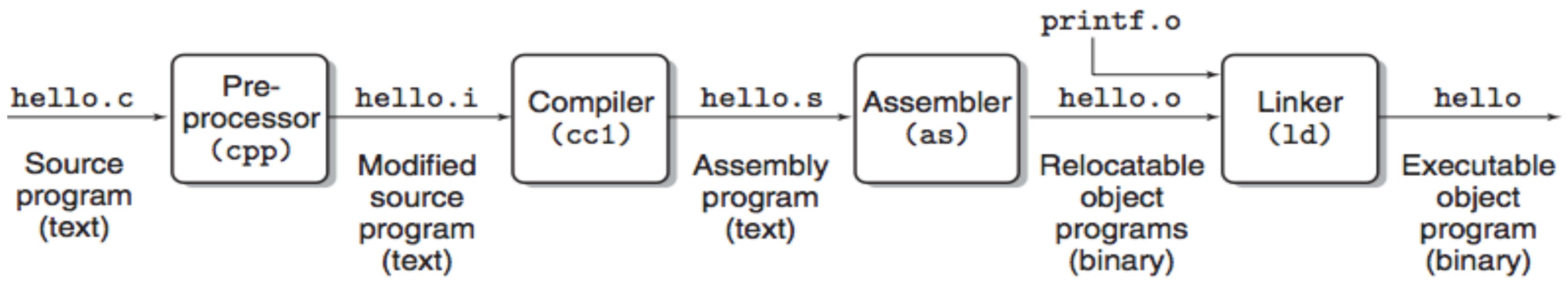
C: Memory leak because it does not free all the nodes

D: Segmentation fault

C Program to a Machine Execution



C Program to a Machine Execution



- What is the interface between the hardware and software?
  - ISA – Instruction Set Architecture
- How to represent information : both data and instructions?
  - How to represent various data types of the programming language?

What Do Computer Do?

Manipulate stored information

Information is data

- How is it represented?

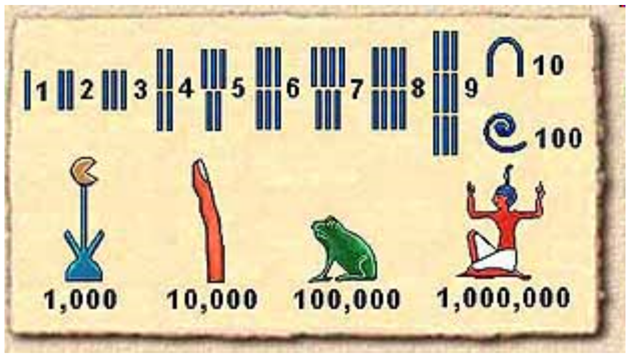
Basic information: numbers

Human beings have represented numbers throughout history

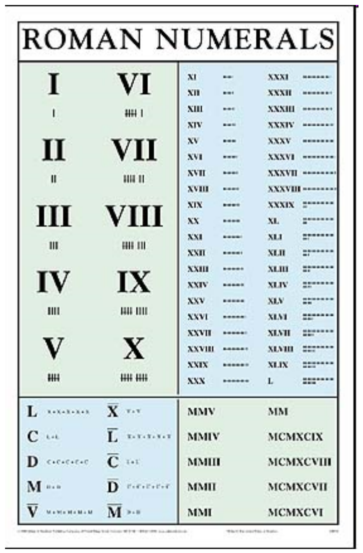
- Egyptian number system
- Roman numeral

Typically decimal

- Natural for humans



Discoveregypt.com



Number System

Comprises of

- Set of numbers or elements
- Operations on them
- Rules that define properties of operations

Need to assign value to numbers

Let us take decimal

- Base 10
- Numbers are written as $d_n ... d_2 d_1 d_0$
- Each digit is in [0-9]
- Value of a number is interpreted as $\sum_{i=0}^{n} d_i \times 10^i$

Binary Numbers

Base 2 $\Rightarrow$ each digit is 0 or 1

Numbers are written as $d_n \ldots d_2 d_1 d_0$

Value of number is computed as $\sum_{i=0}^{n} d_i \times 2^i$

Binary representation is used in computers

- Easy to represent by switches (on/off)
- Manipulation by digital logic in hardware

Written form is long and inconvenient to deal with

Hexadecimal Numbers

Base 16

Each digit can be one of 16 different values

- Symbols = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

First 10 symbols (0 through 9) are same as decimal

- A=10,B=11,C=12, D=13, E=14, F=15

Numbers are written as $d_n \ldots d_2 d_1 d_0$

Value = $\sum_{i=0}^{n} d_i \times 16^i$

Octal Numbers

Base 8 $\Rightarrow$ each digit is in [0-7]

Numbers are written as $d_n \ldots d_2 d_1 d_0$

Value of number is computed as $\sum_{i=0}^{n} d_i \times 8^i$

Converting Hex to Binary

Each hexadecimal digit can be represented by 4 binary digits

- Why?

0x2A8C (hex) = 0b0010101010001100 (binary)

- 0xC = 12 x 16^0 = 8 + 4 = (1 x 2^3) + (1 x 2^2) = 0b1100
- 0x80 = 8 x 16^1 = 2^3 x 2^4 = 2^7 = 0b 10000000
- And so on …

So, to convert hex to binary, just convert each digit and concatenate

What about octal to binary?

Converting Binary to Hex

Do the reverse

- Group each set of 4 digits and change to corresponding digit in hex
- Go from right to left

Example 1011011110011100

- 0b1011011110011100 = 0xB79C

What about binary to octal?

Decimal to Binary

What's the largest p, q, r … such that

- $n = 2^p + r_1$, where $r_1 < 2^p$
- $n - 2^p = 2^q + r_2$, where $r_2 < 2^q$
- $n - (2^p + 2^q) = 2^r + r_3$, where $r_3 < 2^r$
- …

The above means that

- $n = (1 \times 2^p) + (1 \times 2^q) + (1 \times 2^r) + \ldots + r_m$, where $r_m = n \% 2$
- Can you see why this now allows n to be easily written in binary form?

Example: convert 21 to binary

- $21 = 2^4 + 5$, $5 = 2^2 + 1 \Rightarrow 21 = 0b10101$

Decimal to Binary and Back

How to do the conversion algorithmically?

What about binary to decimal?

What about decimal to hex? Hex to decimal?

Decimal to octal? Octal to decimal?

Hex to octal? Octal to Hex?

Decimal and Binary fractions

In decimal, digits to the right of radix point have value $1/10^i$ for each digit in the ith place

- 0.25 is 2/10 + 5/100

Similarly, in binary, digits to the right of radix point have value $1/2^i$ for each ith place

- Just the base is different

8.625 is 1000.101

- .625 = 6/10 + 2/100 + 5/1000 = 1/2 + 1/8

How to convert?

iClicker Pop Quiz I

The binary representation of 66 is

A: 110110

B: 1000010

C: 1111110

D: 10001000

iClicker Pop Quiz II

The hex representation of 66 is

A: 0x66

B: 0x42

C: 1000010

D: 0x96

iClicker Pop Quiz 3

Given a binary representation 011100111, the hexadecimal representation is

A: 0x707

B: 0xe7

C: 0xa70

D: 0x170

iClicker Pop Quiz 4

Given a binary representation 011100111, the octal representation is

A: 0347

B: 0437

C: 0x347

D: 0x123

Decimal to Binary Example

```
Algorithm

number = decimalFraction
while (number > 0) {
    number = number*2
    if (number >=1) {
        Output 1;
        number = number-1
    }
    else {
        Output 0
    }
}
```

Why does it work?

Example: 0.625 to binary

- ANS: 0.101
  - 0.625*2 = 1.25
  - output 1
  - 0.25*2 =0.5
  - output 0
  - 0.5*2 = 1
  - output 1
  - Exit

Data sizes

All Information is represented in binary form but require different sizes

Characters in 1 byte, integers 2 to 4 bytes, real numbers 4 to 8 bytes

C declaration	32-bit machine	64-bit machine
char	1	1
short int	2	2
int	4	4
pointer	4	8
float	4	4
double	8	8

Big Endian vs. Little Endian

How to determine value when have a binary number spread across multiple bytes

A0	**BC**	**00**	**12**

Is it A0BC0012 or 1200BCA0?

- One is called “big endian” and one is “little endian”
  - Most Significant byte the least address … big endian
  - Least significant byte the least address … little endian
- Makes no difference to computer architecture

Why do we care?

- Interpret machine code and values
- One computer (big endian) sending data to another computer (little endian)
- Need to convert into standard form before transmitting

Encoding & Representation

- How do we encode a deck of cards?
  - 13 cards of each suit with 4 such suits
  - Operations: compare the values of the card of the same suit

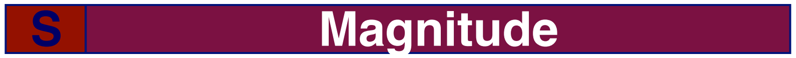
- Any solutions?
  - One shot encoding?

Representing integers

How do we represent negative numbers in computers?

- Use a bit … after all, that's how we store information, right?

Signed Magnitude:



- 0100 = 4, 1100 = -4
- 0011 = 3, 1011 = -3
- Draw a number wheel of the representation

What is 1000?

- Have two zeros +0 (0000) and -0 (1000)
- As we shall see, inconvenient for arithmetic computations
- What is 2 + (-1)?

One’s Complement

Represent negative numbers by complementing positive numbers

Still have two zeros but arithmetic computation becomes easier

000	001	010	011	100	101	110	111
0	1	2	3	-3	-2	-1	-0

Two’s Complement

000	001	010	011	100	101	110	111
0	1	2	3	-4	-3	-2	-1

One’s complement plus one

- What is the intuition? Why does it work? 1 + (-1) = 0
- Most significant bit still gives the “sign”
- Trick: copy all ‘0’ bits from LSB till first ‘1’ bit. Copy ‘1’ bit, then flip all remaining bits till MSB.

Advantages:

- Only 1 zero + convenient for arithmetic computations

Used in almost all computers today

Numerical Value of Two's Complement

What is the maximum value that you can represent with n bits assuming all are natural numbers?

Given a two's complement number of length n, written as $d_{n-1}\ldots d_1 d_0$

- It's value is interpreted as $-d_{n-1}2^{n-1} + \sum_{i=0}^{n-2} d_i 2^i$

The range of values is then $[-2^{n-1}, 2^{n-1} - 1]$

- More negative numbers than positive (if we do not count 0)

2’s Complement Addition & Subtraction

Addition = binary addition, ignore carry out

Subtraction = invert subtrahend and add

```
 -7    1001           -5     1011
+ 5    0101         + -2     1110
----------          ------------
 -2    1110           -7   ~~1~~1001
```

4 – 2 = 4 + -2

```
    4     0100
+  -2     1110
------------
    2     0010
```

2’s Complement Overflow

What happens when overflow occurs with 2’s complement?

- Need one extra bit so sign bit will be wrong

```
   6     0110        -6     1010
+ 5     0101      + -6     1010
-----------       ------------
  -5     1011          4     0100
```

- How to detect?
  - Adding 2 positive numbers ➔ negative result
  - Adding 2 negative numbers ➔ positive result

Multiplication Algorithm

```
  1011
x  101
------
  1011  (multiplicand x 1) shift left 0
 0000   (multiplicand x 0) shift left 1
1011    (multiplicand x 1) shift left 2
------
110111
```

Algorithm

1. result = 0
2. If LSB of multiplier = 1, add multiplicand to result
   else, add 0 to result
3. Shift multiplicand left by 1 bit (fill LSB with 0)
4. Shift multiplier right by 1 bit (fill MSB with 0)
5. If multiplier > 0, go to 1
6. Stop

We only need to know how to add and shift in order to multiply

Signed/Unsigned in C

- Unsigned values in C
  - Declare unsigned int i = 10;
  - Use typecasts i = (unsigned) j;

- -1 when interpreted as a unsigned value is a huge number is
  - $2^n - 1$