Exercise Session 3

Systems Programming and Computer Architecture

Fall Semester 2022

Agenda

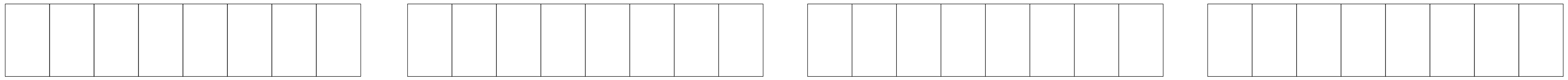
- Last week’s assignment
- More on C-Programing
  - File I/O

Last Week’s Assignment

Bitwise Operations

Bit Lab – bitCount(x)

- Naïve Approach:
  - `(x & 1) + ((x >> 1) & 1) + … + ((x >> 31) & 1)`
  - requires: 31 +, 32 &, 31 >> ▭ 94 operators!!
- Another Idea:
  - Divide 32 bits into segements of «bit counters»



accumumulate bit 25 to 32

accumumulate bit 17 to 24

accumumulate bit 9 to 16

accumumulate bit 1 to 8

Bit Lab - bitCount(x) with counters

- Mask: `unsigned int m = 1 + (1 << 8) + (1 << 16) + (1 << 24)` **6**

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

- Accumulate:

```
unsigned int counts = (x & m) + ((x >> 1) & m) + … ((x >> 7) & m)
```

22

- Sum up:

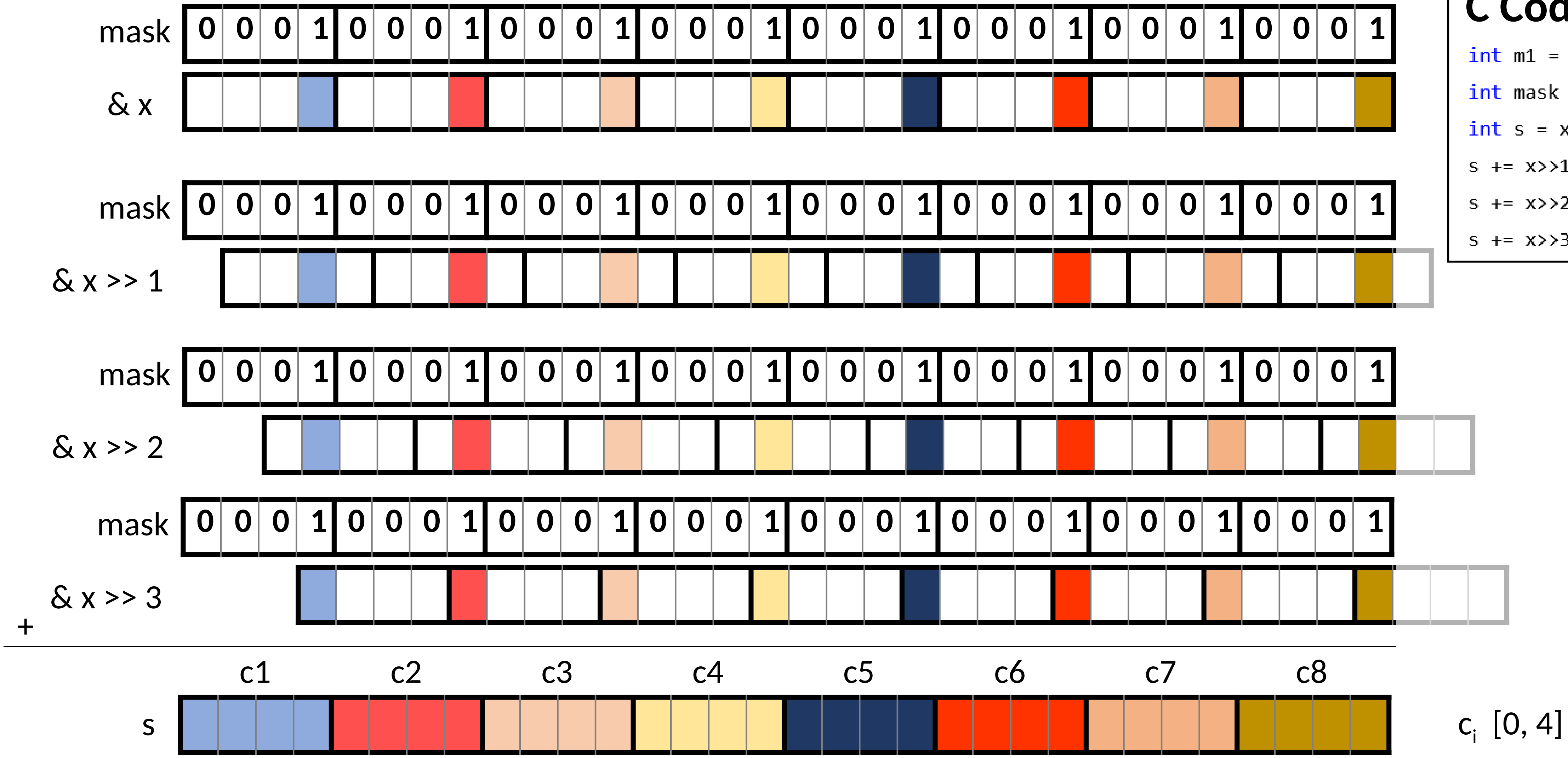
```
unsigned int sum = (counts & 0xFF) + ((counts >> 8) & 0xFF) +
     ((counts >> 16) & 0xFF) + ((counts >> 24) & 0xFF)
```

10

- # Operators? **38**

Bit Lab - bitCount(x) – better solution?

Accumulate in eight 4-bit Counters (rather than 4 8-bit Counters)

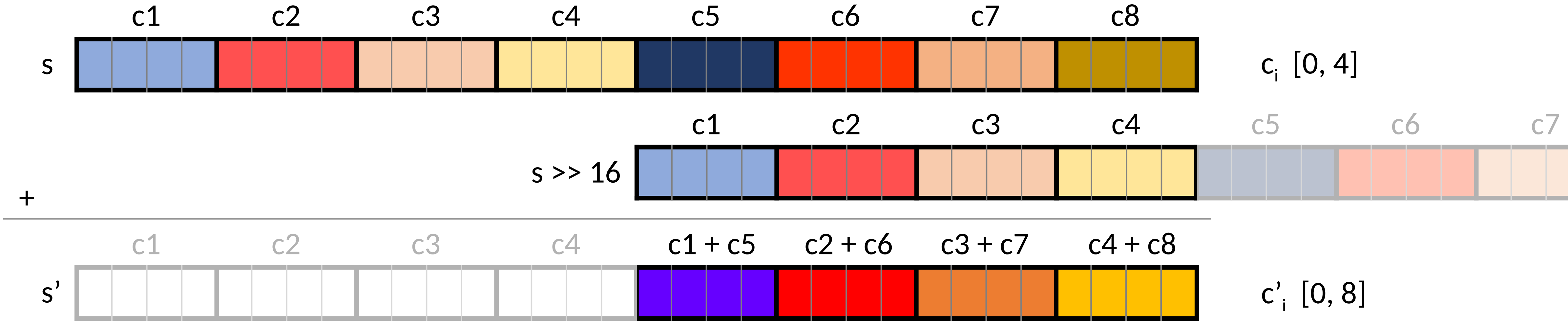


C Code

```
int m1 = 0x11 | (0x11 << 8);
int mask = m1 | (m1 << 16);
int s = x & mask;
s += x>>1 & mask;
s += x>>2 & mask;
s += x>>3 & mask;
```

Step 1: Reduce from 8 to 4 Counters

Combine 4 upper counters with 4 lower counters



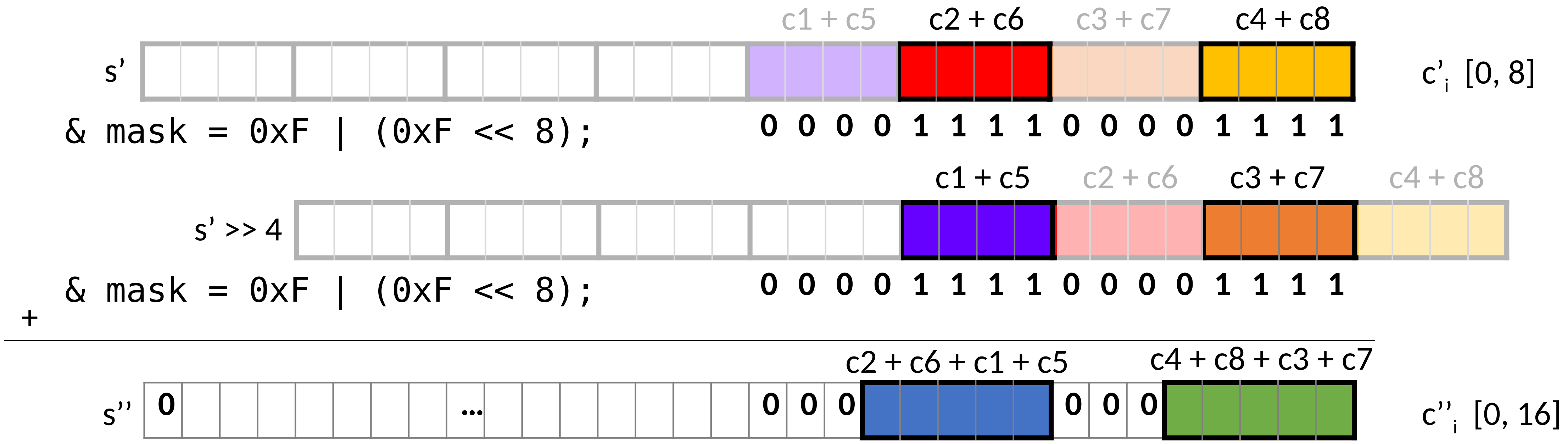
Goal 1 Counter: c1 + c2 + c3 + c4 + c5 + c6

C Code

```
s = s + (s >> 16);
```

Step 2: Reduce from 4 to 2 Counters

Combine 1st with 2nd and 3rd with 4th



c1 + c5 c2 + c6 c3 + c7 c4 + c8

s'

c'$_i$ [0, 8]

```
& mask = 0xF | (0xF << 8);
```

0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1

c1 + c5 c2 + c6 c3 + c7 c4 + c8

s' >> 4

```
& mask = 0xF | (0xF << 8);
```

0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1

+

c2 + c6 + c1 + c5 c4 + c8 + c3 + c7

s'' 0 … 0 0 0 0 0 0

c''$_i$ [0, 16]

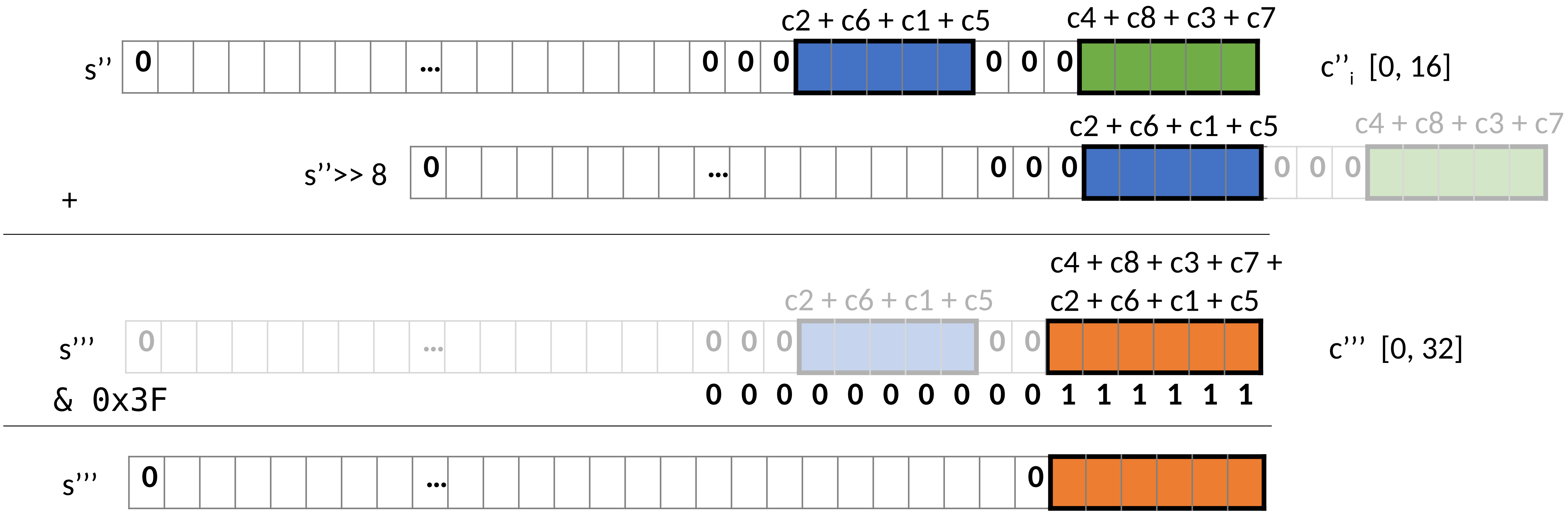
Goal 1 Counter: c1 + c2 + c3 + c4 + c5 + c6

C Code

```
mask = 0xF | (0xF << 8);
s = (s & mask) + ((s >> 4) & mask);
```

Step 3: Reduce from 2 to 1 Counter

Combine the two remaining counters



c2 + c6 + c1 + c5 c4 + c8 + c3 + c7

s'' 0 ... 0 0 0 0 0 0 c''$_i$ [0, 16]

c2 + c6 + c1 + c5 c4 + c8 + c3 + c7

s''>> 8 0 ... 0 0 0 0 0 0

+

c4 + c8 + c3 + c7 +
c2 + c6 + c1 + c5

c2 + c6 + c1 + c5

s''' 0 ... 0 0 0 0 0 c''' [0, 32]

& 0x3F 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1

s''' 0 ... 0

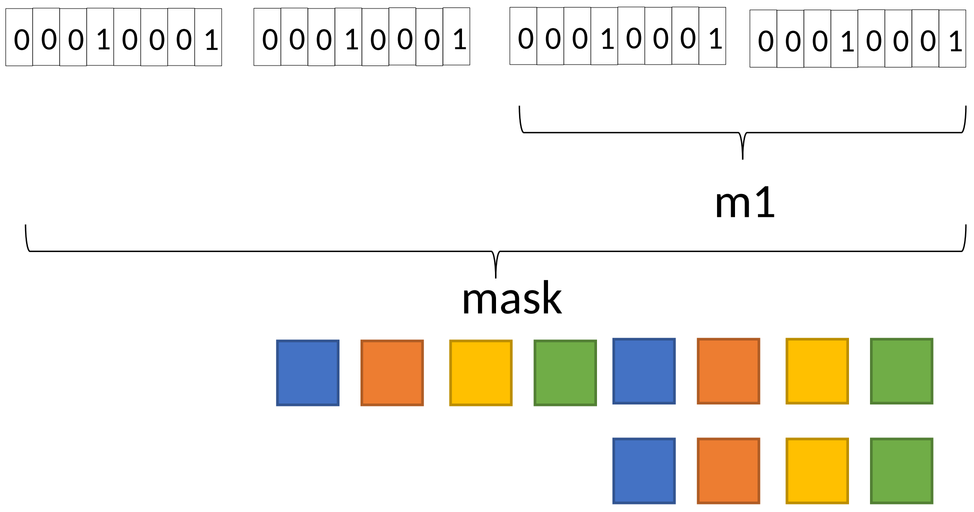
Goal 1 Counter: c1 + c2 + c3 + c4 + c5 + c6

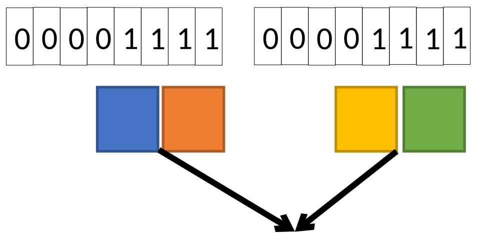
C Code

```c
return (s + (s>>8)) & 0x3F;
```

Bit Lab - bitCount(x) – better solution?

```
int bitCount(int x) {
    /* Sum 8 groups of 4 bits each */
    int m1 = 0x11 | (0x11 << 8);
    int mask = m1 | (m1 << 16);
    int s = x & mask;
    s += x>>1 & mask;
    s += x>>2 & mask;
    s += x>>3 & mask;
    /* Now combine high and low order sums */
    s = s + (s >> 16);
    /* Low order 16 bits now consists of 4 sums,
       each ranging between 0 and 8.
       Split into two groups and sum */
    mask = 0xF | (0xF << 8);
    s = (s & mask) + ((s >> 4) & mask);
    return (s + (s>>8)) & 0x3F;
}
```





Operators: 25

Bit Lab

Questions for Assignment 1?

More on C-Programming

and short overview of assignment 02

Me:
I am good in C language.

Interviewer:
Then write "Hello World" using C.

Me:

```
c   c  ccccc  c      c      ccccc
c   c  c      c      c      c   c
c   c  c      c      c      c   c
ccccc  ccc    c      c      c   c
c   c  c      c      c      c   c
c   c  c      c      c      c   c
c   c  ccccc  ccccc  ccccc  ccccc

c   c  ccccc  ccccc  c      cccc
c   c  c   c  c   c  c      c   c
c c c  c   c  ccccc  c      c   c
cc cc  c   c  cc     c      c   c
c   c  c   c  c c    c      c   c
c   c  c   c  c  c   c      c   c
c   c  ccccc  c   c  ccccc  cccc
```

Integer Data Types

- You may use the types defined in C99’s `stdint.h`

```
#include <stdint.h>

void main(int argc, char *argv[]) {
  uint8_t x = 22;
  int16_t y = 4434;
  int64_t z = 1 << 44;
}
```

- Keep in mind not to overflow by using a too small representation for the value

http://www.cplusplus.com/reference/cstdint/
http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdint.h.html

Reverse an array

Write a C program that has a function that:

- accepts an array of 32-bit unsigned integers and a length
- reverses the elements of the array in place
- returns void (nothing)

```
#include <stdint.h>;
…
void reverse(uint32_t *arr, unsigned int len) {
   …
}
```

Choose Your Types Wisely

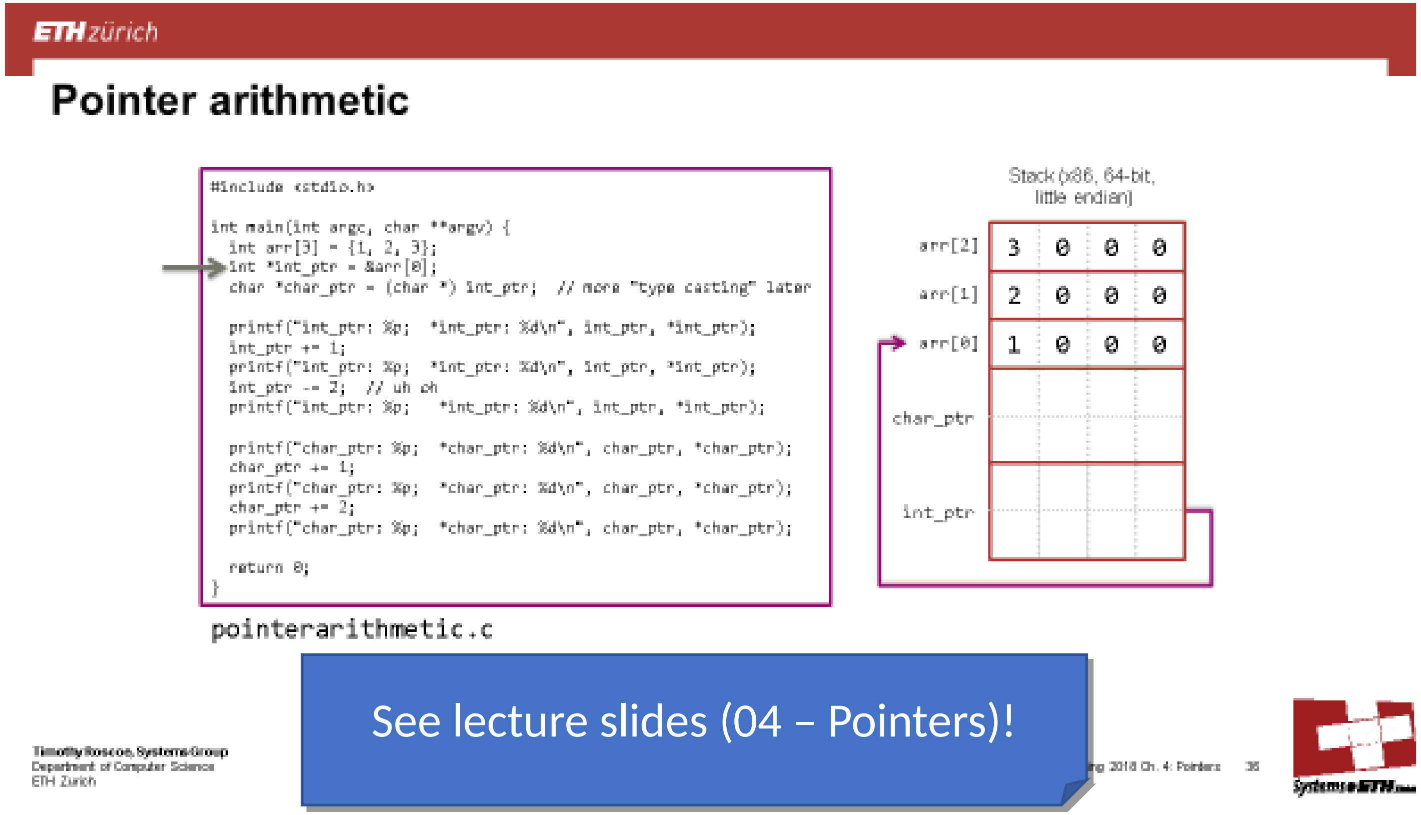
```
void reverse_array(int array[], unsigned int length)
{
  for (int i = 0; i < length; i++) {
    /* … */
  }
}
```

Use the –Wall flag every time !!!!

```
test.cc: In function ‘void reverse_array(int,
unsigned)’:
test.cc:6:22: warning: comparison between signed and
unsigned integer expressions [-Wsign-compare]
    for (int i = 0; i < length; i++) {
```

Box-and-arrow diagram

- Use a box-and-arrow diagram for the given program to explain what it prints out.
- Pen & Paper Exercise: hand it in manually or scan it



ETH zürich

Pointer arithmetic

```
#include <stdio.h>

int main(int argc, char **argv) {
  int arr[3] = {1, 2, 3};
  int *int_ptr = &arr[0];
  char *char_ptr = (char *) int_ptr;  // more "type casting" later

  printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
  int_ptr += 1;
  printf("int_ptr: %p;  *int_ptr: %d\n", int_ptr, *int_ptr);
  int_ptr -= 2;  // uh oh
  printf("int_ptr: %p;   *int_ptr: %d\n", int_ptr, *int_ptr);

  printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
  char_ptr += 1;
  printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);
  char_ptr += 2;
  printf("char_ptr: %p;  *char_ptr: %d\n", char_ptr, *char_ptr);

  return 0;
}
```

pointerarithmetic.c

Stack (x86, 64-bit, little endian)

arr[2]	3	0	0	0
arr[1]	2	0	0	0
arr[0]	1	0	0	0
char_ptr				
int_ptr				

See lecture slides (04 – Pointers)!

Timothy Roscoe, Systems Group
Department of Computer Science
ETH Zürich

ng 2018 Ch. 4: Pointers 36

Little vs. big endian

Write a C program that prints out whether the computer it is running on is little endian or big endian. (hint: pointers and casts)

```
// returns true if little endian, 0 if big endian
bool is_little_endian() {
   …
}
```

Initializing Memory
(First whitespace-separated word)

It’s not safe to assume malloc() returns zeroed memory. So initialize!

```
#include <string.h>

int *data = (int*)malloc(10 * (sizeof(int)));
if (data == NULL) {
  printf(“No memory”);
  exit(1);
}
// ensure memory is zeroed out
memset(data, 0, 10 * sizeof(int)));
```

Remember, use compiler flags

```
gcc <FILES> -Wall -Wpedantic -Wextra -Werror  -std=c99 -Wmissing-prototypes
```

Complex numbers

For this you have to implement a complex number module with following functions:

- add
- subtract
- multiply
- divide

As well as a test driver, which contains the main function.

Dynamic memory allocation

Implement the two functions alloc_set() and free_set(),
but be careful during allocations:

```
struct person * alloc_person() {
   struct person * ret = (struct person *)malloc(sizeof(struct person));
   ret->first_name = (char*)malloc(100 * sizeof(char));
   ret->last_name = (char*)malloc(100 * sizeof(char));
   return ret;
}
```

- malloc() can fail, returns NULL.
- Correct code must not leak memory.

Cleanup code

```c
struct person * alloc_person() {
    struct person * ret = (struct person *)malloc(sizeof(struct person));
    if(!ret)
        return NULL;
    ret->first_name = (char*)malloc(100 * sizeof(char));
    if(!ret->first_name) {
        free(ret);
        return NULL;
    }
    ret->last_name = (char*)malloc(100 * sizeof(char));
    if(!ret->last_name) {
        free(ret->first_name);
        free(ret);
        return NULL;
    }
    return ret;
}
```

WC

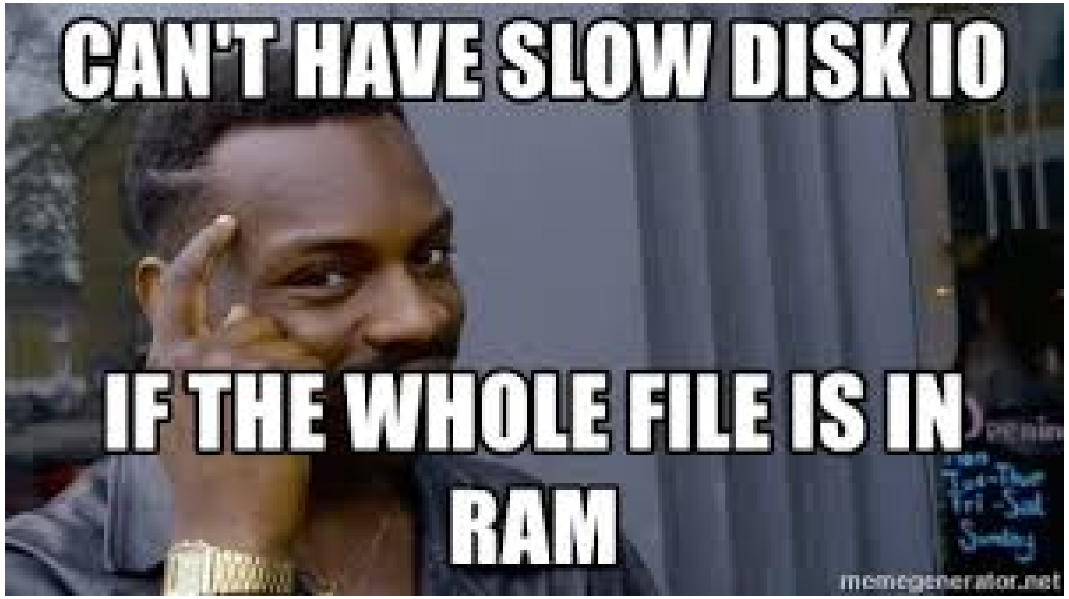
What is wc?

wc is a Unix utility that displays the count of characters, words and lines present in a file.

Implement this unix utility step by step while solving one problem at a time.

Start from the given shell in wc.c and then just add the missing components.

File I/O (File descriptors, read, write)



Accessing Files

- In general you cannot access the file directly

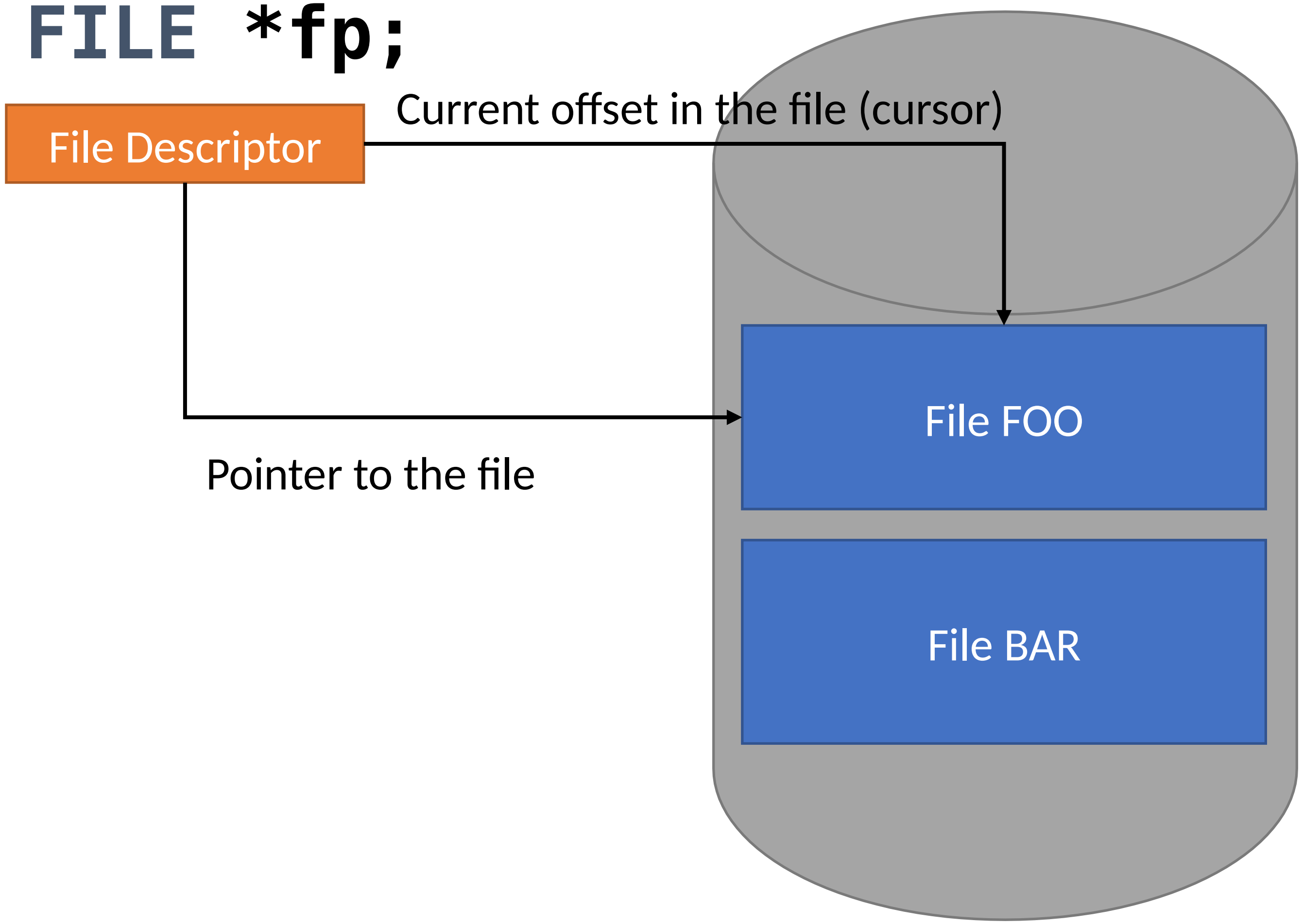
- You need support from the operating system to open/read/write/close a file.
  - This is called a system call (syscall)
    -> Lecture Computer Systems

- All file related declarations are in the stdio.h

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

The File Descriptor

```
FILE *fp;
```



- To get a file descriptor you have to open a file

```
FILE *fopen(const char
*filename,
                const char *mode);
```

- The opening may fail. Check return value!
- Close the file in the end

```
int fclose(FILE *fp);
```

(Text) File Opening Modes

Mode	Read	Write	File Not Exists	File Exists
r	Yes, from beginning	No	Error	FILE* descriptor returned
w	No	Yes, from the beginning	New file created, FILE* Descriptor returned	FILE* descriptor returned
a	No	Yes, from the end (append)	New file crated, FILE* descriptor returned	FILE* descriptor returned
r+	Yes, from beginning	Yes, from beginning	Error	FILE* descriptor returned.
w+	Yes, from beginning	Yes, From beginning	New file created, FILE* descriptor returned	Delete file contents (overwrite), File* descriptor returned
a+	Yes, from beginning	Yes, from the end (append only)	New file crated, FILE* descriptor returned	FILE* descriptor returned
.. b	Binary flag that can be added for binary IO			

(Text) File Opening Modes

Mode	Read	W e	File Not Exists	File Exists
r	Yes, from begin		Error	FILE* descriptor returned
w	No		ew file created, FILE* r returned	FILE* descriptor returned
a	No	Yes, (append)	*	FILE* descriptor returned
r+	Yes, from beginning	Yes, from beginning	Err	FILE* descriptor returned.
w+	Yes, from beginning	Yes, From beginning	New file created, FILE descriptor returned	Delete file contents (overwrite), File* descriptor returned
a+	Yes, from beginning	Yes, from the end (append only)	New file crated, FILE* descriptor returned	FILE* descriptor returned
.. b	Binary flag that can be added for binary IO			

What you (probably) want for the assignment

Reading a Text File (Example)

```
FILE *fp;
int c;
int n = 0;
fp=fopen("myfile.txt","r");
if (fp ==NULL) {
  printf("Error opening
file");
} else {
  do {
    c = fgetc (fp);
    if (c == ‘A') {
      n++;
    }
  } while (c != EOF);
  fclose (fp);
}
printf(“%i“, n);
```

Alternatives…

```
int fgetc(FILE *fp);
char *fgets(char *str, int count, FILE *fp);
size_t fread(void *buf, size_t size,
             size_t count, FILE *fp);
```

Writing a Text File (Example)

```
FILE *fp;
char name];

printf(“Enter your name: ");
fgets (name, 256, stdin);

fp=fopen("myfile.txt",“a");


if (fp ==NULL) {
  printf("Error opening file");
} else {
  if ( fputs (name, fp) < 0) {
    printf("Error writing file");
  }
  fclose (fp);
}
```

Alternatives…

```
int fputc(int character, FILE *fp);

int *fputs(char *str, FILE *fp);

size_t fwrite(void *ptr, size_t size,
              size_t count, FILE *fp);
```

Always keep in mind

- All those functions will change the state of the file descriptor by advancing the cursor position

- Sometimes not all data requested is read/written
  - Keep track of how many bytes are processed
  - Loop until finished

Reading Formatted strings

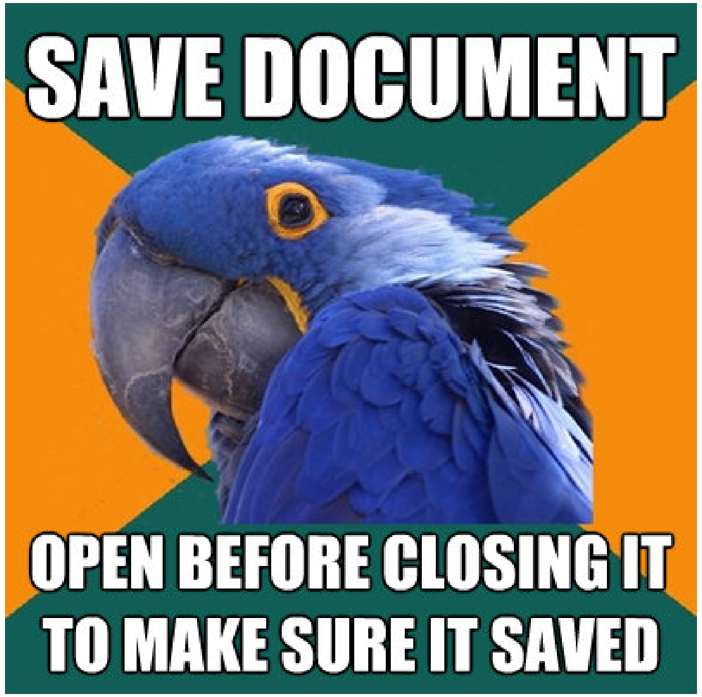
- `int fscanf(FILE *stream, const char *format, ...);`
  - Version of scanf() that reads from given FILE*.
  - `int a; char b[10]; fscanf(stdin, “%d %s”, &a, &b);`
  - Also `sscanf(const char *str, const char *fmt, ...)` may be useful

- Format string:
  - %s Read string
  - %d Read integer

File IO: Resources

- Reference and Examples
  http://www.cplusplus.com/reference/cstdio/

- Read the man pages!
  - man 3 getc
  - man 3 isspace
  - man 3 scanf
- http://stackoverflow.com



Function pointers basics

Write a C program that has a function that:

- accepts a function pointer and an array of integers
- invokes the pointed-to function with each of the elements in the array as an argument
- overrides the current array element with the return value of the called function

```
void map(int (*f) (int), int *array, size_t len) {

}

int pow2(int a) {
  return a*a;
 }
```

Function pointer

This last part will help you get even more familiar with function pointers.

- write a callback fct to use qsort() to sort records based on first name
- write another callback function for last name
- use apply and a callback to filter the records
- BONUS: implement your own mysort()

Short Quiz about pointers

to get you started thinking of pointers

Quiz: Simple Pointer

```
int a[] = { 0,1,2,3,4 };

int i, *p;
for (p = &a[0], i=0; p+i <= a+4; p++,i++)
     printf(“*(p+i) = %d”, *(p+i));
```

Solution: Simple Pointer

- `*p+i = 0  *p+i = 2  *p+i = 4`

Quiz: Arrays and Pointers

```
int a[] = { 0, 1, 2, 3, 4 };
int *p[] = {a, a+1, a+2, a+3, a+4 };
int **pp = p;

main() {
   printf(“…”, a, *a, **a);
   printf(“…”, p, *p, **p);
   printf(“…”, pp, *pp, **pp);
}
```

Quiz: Arrays and Pointers

- a = address of a
  *a = 0
  **a = Segmentation Fault (Null pointer dereference)

- p = address of p
  *p = address of a
  **p = 0

- pp = address of p
  *pp address of a
  **pp = 0

See you
next week!