Levels of Representation/Interpretation

- High Level Language Program (e.g., C)
- Assembly Language Program (e.g., MIPS)
- Machine Language Program (MIPS)

Machine Architecture Description (e.g., block diagrams)
Hardware Architecture Description (e.g., circuit schematic diagrams)

Peer Instruction Question

```c
void main();
{ int *p, x=5, y; // init
  y = *(p = &x) + 1;
  int z;
  flip-sign(p);
  printf("x=%d,y=%d,p=%d\n",x,y,p);
  }
flip-sign(int *n){*n = -*n};
```

How many syntax + logic errors in this C code?

- Red: 1
- Orange: 2
- Green: 3
- Yellow: 4
- Pink: >4

Peer Instruction Answer

```c
#define <stdio.h>
void main()
{ int *p, x=5, y; // init
  y = *(p = &x) + 1;
  int z;
  flip-sign(p);
  printf("x=%d,y=%d,p=%d\n",x,y,p);
  } flip-sign(int *n){*n = -*n};
```

How many syntax + logic errors in this C code?

- Red: 1
- Orange: 2
- Green: 3
- Yellow: 4
- Pink: >4

Agenda

- Arrays
- Administrivia
- Strings
- Pointer Allocation
- Technology Break
- Pointer Problems
- Summary
Arrays (1/5)

- Declaration:
  ```
  int ar[2];
  ```
  declares a 2-element integer array; just a block of memory

- Initializing:
  ```
  int ar[] = {795, 635};
  ```
  declares and initializes a 2-element integer array

- Accessing elements:
  ```
  ar[num]
  ```
  returns the numth element

Arrays (2/5)

- Arrays are (almost) identical to pointers
  - char *string and char string[] are nearly identical declarations
  - Differ in subtle ways: incrementation, declaration of filled arrays

- Key Concept: Array variable is a “pointer” to the first (0th) element

Arrays (3/5)

- Consequences:
  - `ar` is an array variable, but looks like a pointer
  - `ar[0]` is the same as `*ar`
  -尤如的。
  - Array size n; want to access from 0 to n-1, so you should use counter AND utilize a variable for declaration & incrementation
  - Bad pattern
    ```
    int i, ar[10];
    for (i = 0; i < 10; i++) { ... }
    ```
  - Better pattern
    ```
    const int ARRAY_SIZE = 10
    int i, a[ARRAY_SIZE];
    for (i = 0; i < ARRAY_SIZE; i++) { ... }
    ```

- SINGLE SOURCE OF TRUTH
  - You’re utilizing indirection and avoiding maintaining two copies of the number 10

Arrays (4/5)

- Pitfall: An array in C does not know its own length, and its bounds are not checked!
  - Consequence: We can accidentally access off the end of an array
  - Consequence: We must pass the array and its size to any procedure that is going to manipulate it

Array Summary

- Array indexing is syntactic sugar for pointers
  - `a[i]` is treated as `*(a+i)`
  - E.g., three equivalent ways to zero an array:
    ```
    - for (i=0; i < size; i++) a[i] = 0;
    - for (i=0; i < size; i++) *(a+i) = 0;
    - for (p=a; p < a+size; p++) *(p) = 0;
    ```
Agenda

- Arrays
- Administrivia
- Strings
- Pointer Allocation
- Technology Break
- Pointer Problems

1/27/11
Spring 2011 - Lecture #4

### CS 61c in the News

Google is hiring!
And your knowledge of MapReduce is sure to impress them!

Google to hire more than 6,000 workers in 2011

- New search and Speed
- Scribe a Technology career 5/25/11

CHI Staff Writer

May be the biggest hiring year in company’s history.

Google is planning to hire more than 6,000 people this year.

The search engine company told The Associated Press that it would hire more people than it did in 2010, when it added 4,117 workers. In 2011, hiring in Google had increased 4,200.

Google senior vice president of engineering and research, Vic Gundotra said the hiring pace would make this the biggest year for employee growth in company history.

Gundotra said, “At this stage, the number of opportunities is just nearly second the number of people who have at the company.”

The company is dedicated to being aggressive in attracting the more innovation to compete against smaller companies such as Facebook and Twitter.

“The opportunities are a big this year that we have to invest in them and they are how we are going to have to take action and come up to make decisions faster,” Gundotra said.

Google CEO Eric Schmidt was among the business leaders, who met with U.S. President Obama last week to discuss ways to improve the U.S. economy.

The company earned $19 billion last year and ended December with nearly 57 billion in cash.

Agenda

- Arrays
- Administrivia
- Strings
- Pointer Allocation
- Technology Break
- Pointer Problems
- Summary

1/27/11
Spring 2011 - Lecture #4

### Pointers and Structures

```c
struct Point {
    int x;
    int y;
};

Point p1;
Point p2;
Point* paddr;

/* dot notation */
int h = p1.x;
p2.y = p1.y;

/* arrow notation */
int h = paddr->x;
int h = (*paddr).x;

/* This works too */
p1 = p2;
```

1/27/11
Spring 2011 - Lecture #4

### Administrivia

- We have your pictures!
  - file:///Users/randykatz/Documents/Courses/CS61C/Sp11/Photos/cs61Pictures/classpage.html
- This week in lab and homework:
  - Lab #2, MapReduce posted
  - HW #2, Assembly Language posted (hws are leading the lecture – this is a good forcing function to read ahead)
- Join the class discussion/announcements group!
  - But don’t forget that some stuff is VERY easy to find on your own (e.g., arrow notation in C/C++)

1/27/11
Spring 2011 - Lecture #4

### Pointers

- Arrays
- Administrivia
- Strings
- Pointer Allocation
- Technology Break
- Pointer Problems
- Summary
C Strings
• String in C is just an array of characters
  
  char string[] = "abc";
• How do you tell how long a string is?
  – Last character is followed by a 0 byte
    (aka “null terminator”)
    int strlen(char s[])
    {
      int n = 0;
      while (s[n] != 0) n++;
      return n;
    }

Pointer Arithmetic

Array + number  Pointer – number
E.g., pointer + 1 adds 1 something to a pointer

    char *p;
    char a;
    char b;
    p = &a;
    p += 1;

In each, p now points to b
(Assuming compiler doesn’t reorder variables in memory)

Add 1*sizeof(char) to the memory address
Add 1*sizeof(int) to the memory address

Pointer arithmetic should be used carefully

Arrays and Pointers
• An array is passed to a function as a pointer
  – The array size is lost!
• Usually bad style to interchange arrays and pointers
  – Avoid pointer arithmetic!

```c
int foo(int array[], unsigned int size)
{
  for (i = 0; i < size; i++)
    printf("%d", array[i]);
}
int main(void)
{
  int a[10], b[5];
  foo(a, 10);
  foo(b, 5);
}
```

What does this print? 8
... because array is really a pointer and array is architecture dependent, but likely to be 8 on modern machines

What does this print? 40

Arrays and Pointers
• Array = pointer to the initial (0th) array element
  
  a[1] = *(a+1)
• An array is passed to a function as a pointer
  – The array size is lost!
• Usually bad style to interchange arrays and pointers
  – Avoid pointer arithmetic!

```c
int i;
int array[10];
for (i = 0; i < 10; i++)
  { 
    array[i] = ...
  }
```

These code sequences have the same effect!

Arrays and Pointers (1/2)
• Since a pointer is just a memory address, we can add to it to step through an array
  
  p+1 correctly computes a ptr to the next array element automatically depending on sizeof(type)
  
  *p++ vs. (*p)++
  
  x = *p++  ⇒  x = *p;  p = p + 1;
  x = (*p)++  ⇒  x = *p;  *p = *p + 1;
  
  This is a C syntax/semanitics thing
• What if we have an array of large structs (objects)?
  – C takes care of it in the same way it handles arrays
### Pointers & Allocation (1/2)

- After declaring a pointer:
  - `int *ptr;`
- `ptr` doesn’t actually point to anything yet (points somewhere, but don’t know where).
  - We can either:
    - Make it point to something that already exists, or
    - Allocate room in memory for something new that it will point to...

- Allocate room in memory for something new that
  - Make it point to something that already exists, or
  - Allocate room in memory for something new that it will point to...

### Peer Instruction

```c
int main(void){
  int *p = A;
  printf("%u %d %d \n", p, p, A[0], A[1]);
  p = &p[1];
  printf("%u %d %d \n", p, p, A[0], A[1]);
  p = &p[1];
  printf("%u %d %d \n", p, p, A[0], A[1]);
}
```

- If the first printf outputs 100 5 10, what will the next two printf output?
  - Red a) 101 10 5 10 then 104 11 11
  - Orange b) 104 10 5 10 then 104 11 11
  - Green c) 104 other 5 10 then 104 other 11
  - Yellow d) 104 other 5 10 then 104 other 11
  - Pink e) One of the two prints causes an ERROR

- `var1` and `var2` have space implicitly allocated for them

### Agenda

- Arrays and Strings
- Administrator
- Pointer Allocation
- Technology Break
- Pointer Problems
- Summary
Arrays
(one element past array must be valid)
• Array size n: want to access from 0 to n-1, but test for exit by comparing to address one element past the array

```c
int ar[10], *p, *q, sum = 0;
...
p = &ar[0]; q = &ar[10];
while (p != q):
    /* sum = sum + *p; p = p + 1; */
    sum += *p++;
Is this legal?
• C defines that one element past end of array must be a valid address, i.e., will not cause an bus error or address error
```

Pointer Arithmetic
• What is valid pointer arithmetic?
  – Add an integer to a pointer
  – Subtract 2 pointers (in the same array)
  – Compare pointers (<, <=, ==, !=, >, >>)
    – Compare pointer to NULL (indicates that the pointer points to nothing)
  • Everything else is illegal since it makes no sense:
    – Adding two pointers
    – Multiplying pointers
    – Subtract pointer from integer

Pointer Arithmetic to Copy Memory
• We can use pointer arithmetic to "walk" through memory:

```c
void copy(int *from, int *to, int n)
{
    int i;
    for (i=0; i<n; i++) {
        *to++ = *from++;
    }
}
• Note we had to pass size (n) to copy
```

Pointer Arithmetic Summary
• x = *(p+1) ?
  → x = *(p+1);
• x = *p+1 ?
  → x = (*p) + 1;
• x = *(p++)?
  → x = *(p++);
• x = *p++ ? *(p++) ? *(p++) ? *(p++) ?
  → x = *p; p = p + 1;
• x = ++p ?
  → p = p + 1;
• Lesson?
  – Using anything but the standard *p++, *(p++) causes more problems than it solves!

Pointer Arithmetic: Peer Instruction Question
How many of the following are invalid?
I. pointer + integer
II. integer + pointer
III. pointer + pointer
IV. pointer – integer
V. integer – pointer
VI. pointer – pointer
VII. compare pointer to pointer
VIII. compare pointer to integer
IX. compare pointer to 0
X. compare pointer to NULL

#invalid
Red: 1
Orange: 2
Green: 3
Yellow: 4
Pink: 5

Pointer Arithmetic: Peer Instruction Answer
How many of the following are invalid?
I. pointer + integer
II. integer + pointer
III. pointer + pointer
IV. pointer – integer
V. integer – pointer
VI. pointer – pointer
VII. compare pointer to pointer
VIII. compare pointer to integer
IX. compare pointer to 0
X. compare pointer to NULL

#invalid
Red: 1
Orange: 2
Green: 3
Yellow: 4
Pink: 5
C Functions

- Functions: How to structure C programs for understandability and reuse
- Calling function: `s = max(x, y, 100);`
- If no value to return, declare it type `void`
  - Function with no return value called `procedure` in other programming languages

Calling function:
```c
int s = max(x, y, 100);
```

If no value to return, declare it type `void`
- `FuncSon` with no return value called `procedure` in other programming languages

Give name of `FuncSon` and type of value it returns
```c
give name of funcSon and type of value it returns
```

```c
int max(a, b, c) /* declaration */
int a, b, c; /* type of params */
{
    int m;
    m = (a > b) ? a : b;
    return (m > c ? m : c);
}
```

Pointers and Functions (1/4)

- Sometimes you want to have a procedure increment a variable ...
- What gets printed?
  ```c
  void AddOne(int x)
  {
      x = x + 1;
  }
  int y = 5;
  AddOne(y);
  printf("y = %d\n", y);
  ```

Pointers and Functions (2/4)

- Solved by passing in a `pointer` to our subroutine.
- Now what gets printed?
  ```c
  void AddOne(int *p)
  {
      *p = *p + 1;
  }
  int y = 5;
  AddOne(&y);
  printf("y = %d\n", y);
  ```

Pointers and Functions (3/4)

- But what if the thing you want changed is a `pointer`?
- What gets printed?
  ```c
  void IncrementPtr(int *p)
  {
      *p = *p + 1;
  }
  int A[3] = {50, 60, 70};
  int *q = A;
  IncrementPtr(q);
  printf("*q = %d\n", *q);
  ```

Pointers and Functions (4/4)

- Solution! Pass a `pointer to a pointer`, declared as `**h`
- Now what gets printed?
  ```c
  void IncrementPtr(int **h)
  {
      **h = **h + 1;
  }
  int A[3] = {50, 60, 70};
  int *q = A;
  IncrementPtr(&q);
  printf("*q = %d\n", *q);
  ```
C String Standard Functions

#include <string.h>

- int strlen(char *string);
  - Compute the length of string
- int strcmp(char *str1, char *str2);
  - Return 0 if str1 and str2 are identical (how is this different from str1 == str2?)
- char *strcpy(char *dst, char *src);
  - Copy contents of string src to the memory at dst. Caller must ensure that dst has enough memory to hold the data to be copied
  - Note: dst = src only copies pointers, not string itself

Segmentation Fault vs. Bus Error

- http://www.hyperdictionary.com/
- Bus Error
  - A fatal failure in the execution of a machine language instruction resulting from the processor detecting an anomalous condition on its bus. Such conditions include invalid address alignment (accessing a multi-byte number at an odd address), accessing a physical address that does not correspond to any device, or some other device-specific hardware error. A bus error triggers a processor-level exception which Unix translates into a "SIGBUS" signal which, if not caught, will terminate the current process.
- Segmentation Fault
  - An error in which a running Unix program attempts to access memory not allocated to it and terminates with a segmentation violation error and usually a core dump.

C Pointer Dangers

- Unlike Java, C lets you cast a value of any type to any other type without performing any checking
  ```c
  int x = 1000;
  int *p = x; /* invalid */
  int *q = (int *) x; /* valid */
  ```
- First pointer declaration is invalid since the types do not match
- Second declaration is valid C but is almost certainly wrong
  - Is it ever correct?

C String Problems

- Common mistake is to forget to allocate an extra byte for the null terminator
- More generally, C requires the programmer to manage memory manually (unlike Java or C++)
  - When creating a long string by concatenating several smaller strings, the programmer must insure there is enough space to store the full string!
  - What if you don’t know ahead of time how big your string will be?
  - Buffer overrun security holes!
More Common C Errors

• Difference between assignment and equality
  
  \[ a = b \] is assignment
  \[ a == b \] is an equality test

• One of the most common errors for beginning C programmers!
  – One pattern (when comparing with constant) is to put the var on the right!
  If you happen to use =, it won’t compile!
    • \[ if (3 == a) \{ ... \} \]

And in Conclusion, ...

• Pointers are aliases to variables
• Pointers can be used to index into arrays
• Strings are (null terminated) arrays of characters
• Pointers are the source of many bugs in C, so handle with care