CS61C
Machine Structures
Lecture 2

September 1,2000

Dave Patterson
(http.cs.berkeley.edu/~patterson)

http://www-inst.eecs.berkeley.edu/~cs61c/

Review 1/2

° 15 weeks to learn big ideas in CS&E
- Principle of abstraction, used to build systems as layers
- Pliable Data: a program determines what it is
- Stored program concept: instructions are just data
- Principle of Locality, exploited via a memory hierarchy (cache)
- Greater performance by exploiting parallelism (pipeline)
- Compilation v. interpretation to move down layers of system
- Principles/Pitfalls of Performance Measurement

Review 2/.2

° Continued rapid improvement in Computing
- 2X every 1.5 years in processor speed; every 2.0 years in memory size; every 1.0 year in disk capacity; Moore's Law enables processor, memory (2X transistors/chip/ ~1.5 yrs)

° 5 classic components of all computers

$\underbrace{\text{Control} \quad \text{Datapath}}_{\text{Processor}} \quad \text{Memory} \quad \text{Input} \quad \text{Output}$

Overview

° Review (2 minutes)

° C Pointers (10 min)

° Parameter Passing (8 min)

° Administrivia Break (5 min)

° Computers in the News (3 min)

° C structures (10)

° Memory Allocation (10 min)

° Conclusion (2 min)

C vs. Java - Differences

° C has no objects, no classes, no superclasses
- Java method => C procedure, function

° Variable, Array initialization
- C: ? (sometimes zero, sometimes random)
- Java: zero

° Output
- C: `printf()`
- Java: `system.out.print()`

° Memory management, pointers

C Pointers

° Pointer: represents a raw memory address (more details later in course)

° C variables can have pointer types:

```
int *x;  /* type: pointer to an int */
int **x; /* type: pointer to a pointer to an int */
int *x, *y, *z;  /* type: pointers to ints */
int *x, y, z;  /* type: ? */
```

° How create a pointer to store in a variable?

¥& operator: get address of a variable

¡`int *x;` /* type: pointer to an int */

C Pointers

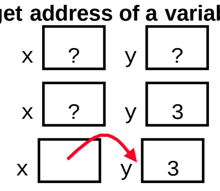
° How create a pointer to store in a variable?

¥& operator: get address of a variable

```
int *x, y;

   y = 3;

  y = &x;
```



° How get a value pointed to?

¥* dereference operator: get value pointed to

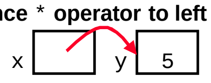
```
printf( x points to %d\n ,*x);
```

C Pointers

° How change variable pointed to?

- Use dereference * operator to left of =

```
*x = 5;
```



° How get a address of the pointer?

- Why do you want to do this???

```
printf( x pointer is %p\n , x);
```

° Closest thing in C to object type in Java:

¥void * is a type that can point to anything

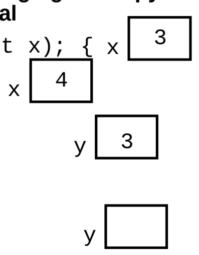
- Example: linked list in C with pointer to next node and a pointer to the value, but not sure what type it is

C Pointers and Parameter Passing

° Java and C pass a parameter "by value"

- procedure/function gets a copy of the parameter, so changing the copy cannot change the original

```
void addOne (int x); {
   x = x + 1;
};
int y = 3;
addOne (y);
```

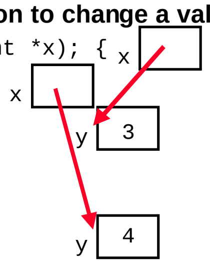


C Pointers and Parameter Passing

° How get a function to change a value?

```
void addOne (int *x); {
   *x = *x + 1;
};
int y = 3;

addOne (&y);
```



Administrivia 1/4

° Change in TAs, new people, shuffle sections (you asked for it)

° Students who have not taken 61B:

- Will be dropped from class if enrolled or not promoted from wait list

° If you have taken 61B or the equivalent and you are on the list:

- Petition Michael-David Sasson, 379 Soda, 643-6002, msasson@cs to petition today

° 61B Fall Semester meets in the same room, so it can easily add 100 people; more sections will be added as needed

Administrivia 2/4

° Instructor: David A. Patterson (patterson@cs) 635 Soda Hall
Office Hours:Wed 1-2

- Sally Mack (smack@cs), 626 Soda, x3-4034

° Materials: http://www-inst.eecs/~cs61c

° Lectures, projects, labs, homeworks on www-inst.eecs/~cs61c/schedule.html

° Newsgroup: ucb.class.cs61c

° Text: Computer Organization and Design: The Hardware/Software Interface, 2/e

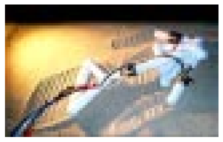
Administrivia 3/4

- Must turn in survey, login and attend lab/discussion sections to be considered enrolled
  - Go to old and new sections to ask TA's to switch sections
- Lab exercises are to be done every week in lab section, and checked off by your lab TA or turned in at beginning of lab
- Homework exercises are to be handed in either online or to homework boxes in 283 Soda, due on Mondays at noon;
- Projects are larger programming assignments; individual and team

Administrivia 4/4

- Course Exams
  - Midterm: Wednesday October 25 (5-8 PM, 1 Pimentel)
  - Final: Tuesday, December 12 (5-8 PM, 1 Pimentel)
- Class agreed upon punishment for
  - Cheating on Exercises, Labs, Projects
    - 0 for assignment
  - Cheating on Exams
    - 0 for exam

Computers in the News


www.wired.com/wired/archive/8.04/joy.html
golem03.cs-i.brandeis.edu/download/AutomaticDesign.pdf

- "Artificial Life Milestone: Robots Building Robots"

For the first time, computer scientists have created a robot that designs and builds other robots, almost entirely without human help. In the short run, this advance could lead to a new industry of inexpensive robots customized for specific tasks. ...

Mimicking biological evolution, the computer added, subtracted and changed pieces in the designs. At the same time, the computer similarly mutated the programming instructions for controlling the robot's movements. After each step, the computer ran simulations to test the designs, keeping the ones that moved well and discarding the failures.

"Some thing we probably can do we shouldn't do", Bill Joy... "We're on the road to somewhere where there's big issues..."

N.Y. Times, front page, 8/31/00

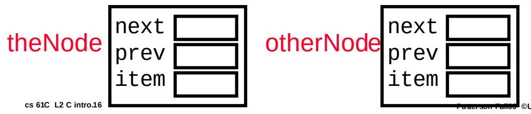
C Structures vs. Java Classes

- Structure: Agglomerations of data;like a Java class, but no methods attached

```
struct DlistNode {
    struct DlistNode *next;
    struct DlistNode *prev;
    void *item;
}; /* need semicolon in C */
```

- Now create DlistNode variables

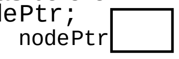
```
struct DlistNode theNode;
struct DlistNode otherNode;
```



C Structures vs. Java Classes

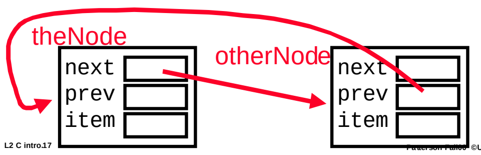
- Create pointer to structure as before

```
struct DlistNode * nodePtr;
```



- Now can assign to fields

```
theNode.next = &otherNode;
otherNode.prev = &theNode;
```



C Memory Management

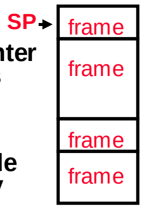
- C requires knowing where objects are in memory, otherwise don't work as expect
  - Java hides location of objects
- C has 3 pools of memory
  - Static storage: global variable storage, basically permanent, entire program run; not in Java (easier to combine, no side effects, re-entry easier)
  - The Stack: local variable storage, debugging info, parameters, return address (location of "activation records" in Java or "stack frame" in C)
  - The Heap (dynamic storage): data lives until deallocated by programmer

The Stack

° Stack frame includes:
  • Return address
  • Parameters
  • Space for other local variables

° Stack frames contiguous blocks of memory; stack pointer tells where top stack frame is

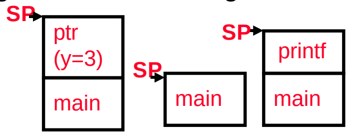
° When procedure ends, stack frame is tossed off the stack, using return address to decide where to go to; frees memory for future stack frames



Who cares about stack management?

° Pointers in C allow access to deallocated memory, leading to hard to find bugs !

```
int * ptr () {
   int y;
   y = 3;
   return &y;
 };
 main () {
   int *stackAddr,content;
   stackAddr = ptr();
   content = *stackAddr;
   printf("%d", content); /* 3 */
   content = *stackAddr;
   printf("%d", content); /*13451514 */
 };
```



The Heap (Dynamic memory)

° Large pool of memory, not allocated in contiguous order
  • back-to-back requests for heap memory could result blocks very far apart
  • where Java `new` command allocates memory

° In C, specify number of bytes of memory explicitly to allocate item

```
int *ptr;
 ptr = (int *) malloc(4);
   /* malloc returns type void *, so
 need to cast to right type */
```

¥`malloc`: Allocates raw, uninitialized memory from heap

C Memory Allocation: `malloc()`

° Instead of explicit number, for portability, use `sizeof()`

```
int *ptr;
 ptr = (int *) malloc(sizeof(int));
```

  • not a procedure; will check type or a variable to turn into a number

° malloc() also for structure allocation

```
struct DlistNode * nodePtr;
 nodePtr = (struct DlistNode *)
   malloc(sizeof(struct DlistNode));
```

  • Note: unlike Java, C never frees memory; programmer must explicitly free memory

C Memory Allocation

° Rule of thumb: deallocate anything you're never going to use again
  • If not too much, and program doesn't run a long time, then allocate a lot at the beginning and then let memory be freed when program ends
  • Otherwise, end up with "Memory Leaks", that is, program gets bigger over time, and need to restart computer

¡`free()` is opposite of `malloc()`
  • Danger: may accidentally deallocate memory when still have a pointer into it, causing the same problem as with pointers to stacks

Odds and Ends

° Structure declaration does not allocate memory

° Variable declaration does allocate memory
  • If declare inside procedure, allocated on the stack
  • If declare outside a procedure, allocated in static storage

"And in Conclusion…"

- C is an efficient language, with little protection
  - Array bounds not checked
  - Variables not automatically initialized
- C v. Java: pointers and explicit memory allocation and deallocation
  - No garbage collection
  - Leads to memory leaks, funny pointers
  - Structure declaration does not allocate memory; use `malloc()` and `free()`
- Designed for writing systems code, device drivers