CS106X
Spring 99-00

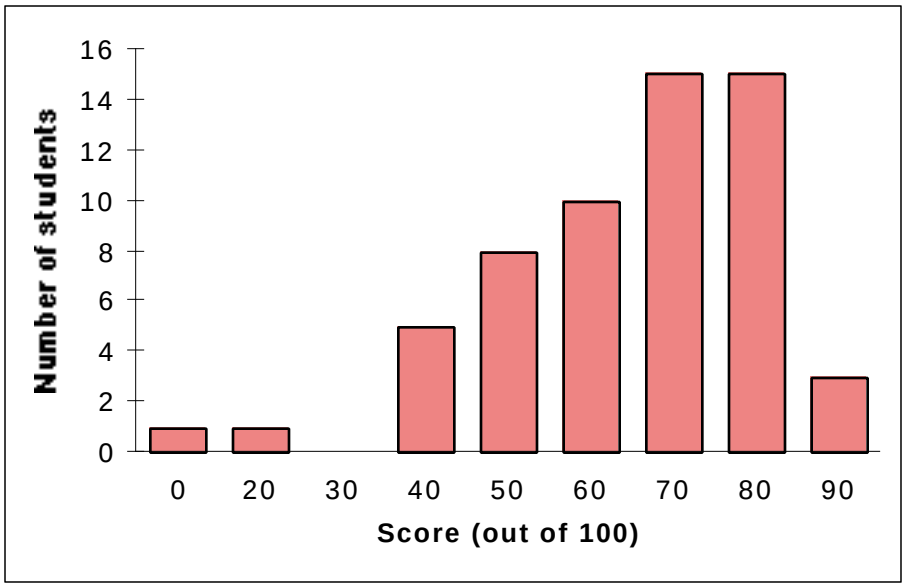
Handout #30
May 3, 2000

Midterm Solutions

The midterms have all been graded and we'll hand them back at the end of Wednesday's lecture. We had almost as much fun grading them as you did taking them :-)

You all agreed how tough/long it was, but, in fact, your answers didn't show that you had much trouble with it. The class as a whole did very well and most of you clearly have this material down cold. The most common mistakes were minor ones that you would have easily corrected with a little more time to go over your work. I was pleased and impressed!

The median was 73, the mean 68, and the standard deviation was 16. In accordance with standard Stanford Grade Inflation™ policy, I set the mapping such that the median is dividing line between the highest B and the lowest B+ and about 40% of the students will receive some form of A, another 40% some form of B (106X is an intense class and self-selected for motivated students, so it seems perfectly reasonable to give a lot of good grades and I'm all for it!)



Regrades: if you believe there was an error in scoring your exam, please submit it to Andrew or me within a week for a correction. The entire exam will be re-evaluated for any discrepancies. You must return the grading sheet with your exam.

Problem 1: Encode

A solution using array notation:

```
static string Encode(string s)
{
   string result = GetBlock(strlen(s) + 1);
   int i, pos, count;

   pos = 0;  // tracks position to write in result string
   for (i = 0; s[i] != '\0'; i++) {
       for (count = 1; s[i + count] == s[i]; count++)
           ;
       if (count > 1) {
           result[pos++] = count + '0';
           i += count - 1; // -1 because of for loop i++
       }
       result[pos++] = s[i];
   }
   result[pos] ='\0';
   return result;
}
```

An alternative using pointers:

```
static string Encode(string s)
{
   char *result = GetBlock(strlen(s) + 1);
   char *out = result;

   while (*s != '\0') {
        int count = 1;
       while (*s == *(s + count))  // will stop at null char, too
          count++;
       if (count > 1)
           *out++ = count + '0';
       *out++ = *s;
       s += count;
   }
   *out ='\0';
   return result;
}
```

Grading (mean for this question: 12/16)

Answers to this question were quite good. Students tended to want to write the solution using pointer arithmetic, which surprised me, given how nasty and complicated it usually works out in that syntax. Don't forget you can use array notation— especially when it is more clear or convenient!

Problem 2: InsertAtPosition

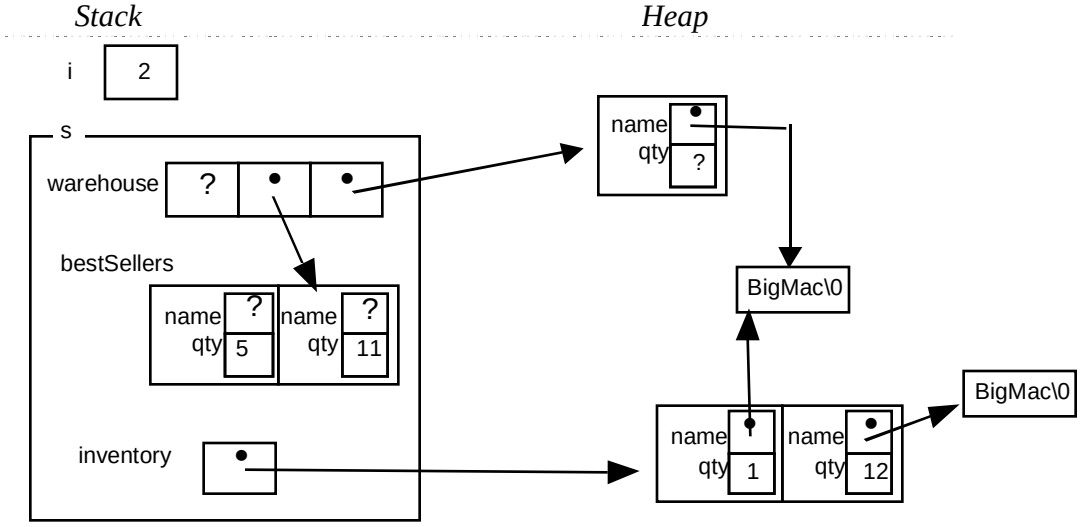
```
static bool InsertAtPosition(Cell **head, double value, int position)
{
     int i;
     Cell *prev = NULL, *cur = *head, *newOne;

     if (position < 0)
         return FALSE;
     for (i = 0; i < position; i++) {
        if (cur == NULL) return FALSE;
        prev = cur;
        cur = cur->next;
     }
       // at this point, prev is n-1, cur is nth, insert new inbetween
     newOne = New(Cell *);
     newOne->value = value;
     if (prev) {
        newOne->next = prev->next;
        prev->next = newOne;
     } else {
        newOne->next = *head;
        *head = newOne;
     }
     return TRUE;
}
```

A recursive alternative, for those already psyched about the things to come:

```
static bool RecInsert(Cell **head, double value, int position)
{
     if (position < 0)
         return FALSE;
     if (position == 0) {
         Cell *newOne = New(Cell *);
         newOne->value = value;
         newOne->next = *head;
         *head = newOne;
         return TRUE;
     } else {
        return (*head != NULL &&
                RecInsert(&(*head)->next, value, position -1);
    }
 }
```

Grading (mean for this question: 14/18)
There were various special cases to consider (insert at head, insert into empty list, running off the end, etc.) that made this one a tough one to get all the details right, but most of you did very well despite all the pitfalls.

Problem 3: Pointers and program tracing



Grading (mean for this question: 12/16)
You guys sure know your pointers!

Problem 4: Data structures

```
Event *
Member *
string * (or char **)
chessClub.calendar[15].events[0]->time // or day index 14
chessClub.members[5]->calendar[3].nEvents // or day index 2


static bool IsAvailable(Day calendar[], int day, int time)
{
   int i;

   if (calendar[day].nEvents == MAX_EVENTS_PER_DAY)
      return FALSE; // day is full
   for (i = 0; i < calendar[day].nEvents; i++) {
      if (calendar[day].events[i]->time == time)
          return FALSE;
   }
   return TRUE;
}


static int CountAttendees(Club *club, int day, int time)
{
   int i, count;

   count = 0;
   for (i = 0; i < club->nMembers; i++)      // check all members
      if (IsAvailable(club->members[i]->calendar, day, time))
          count++;
   return count;
}
```

```
static void ScheduleEvent(Club *club, string eventName, int time)
{
   int day, count, bestCount;
   Day *bestDay = NULL;
   Event *ev;

   bestCount = 0;
   for (day = 0; day < NUM_DAYS; day++) {
      if (IsAvailable(club->calendar, day, time)) { // check club cal
          count = CountAttendees(club, day, time);
          if (count >= bestCount) {
              bestCount = count;
              bestDay = &club->calendar[day];
          }
      }
   }
   ev = New(Event *);
   ev->name = CopyString(eventName);
   ev->time = time;
   bestDay->events[bestDay->nEvents++] = ev;
}
```

Grading (mean for this question: 26/32)
Throughout #4 and #5, we accepted both day and day +/- 1 interpretations as full-credit (and in fact, didn't even check if you were consistent about it, shh…). The most common errors were forgetting to allocating the new Event record and some confusions accessing the pieces of the data structure. Clearly work on Battleship paid off on this question.

Problem 5: File processing

```
static Club *ReadOneClub(FILE *f)
{
   string name;
   Club *club;
   int i, day, numEvents;

   name = ReadLine(f);
   if (name == NULL) return NULL;      // check for EOF

   club = New(Club *);
   club->name = name;
   for (i = 0; i < NUM_DAYS; i++)
      club->calendar[i].nEvents = 0; // init all days to no events

   fscanf(f, "Events: %d\n", &numEvents);
   for (i = 0; i < numEvents; i++) {
      Event *ev = New(Event *);
      fscanf(f,"Day: %d Time: %d\n", &day, &ev->time);
      ev->name = ReadLine(f);
      club->calendar[day].events[club->calendar[day].nEvents++] = ev;
   }
   return club;
}
```

Grading (mean for this question: 11/18)
Most common errors were forgetting about the EOF check and failing to initialize the calendar to show no events. This was also a lot like Battleship, so familiar territory.