CS106X
Spring 99-00

Handout #22
April 24, 2000

Practice Solutions

Midterm: Monday, May 1st 7-9 pm
Gates B01(next door to our usual room)

Problem 1: Strings Revealed

There are many different approaches to solving this problem, two are presented here. The first uses a loop to do the alternating weave until one of the strings runs out, at which point it calls strcpy to dump the remaining contents onto the result string. It manipulates the strings with array subscript notation.

```
   char *Weave1(char s1[], char s2[])
   {
      int i, len1 = strlen(s1), len2 = strlen(s2);
      char *result;

      result = (char *)GetBlock(len1 + len2 + 1);
      for (i = 0; i < len1 && i < len2; i++) {
         result[2*i] = s1[i];
         result[2*i + 1] = s2[i];
      }
      if (len1 < len2)
         strcpy(result + 2*i, s2 + i);    // strcpy appends null char
      else
         strcpy(result + 2*i, s1 + i);
      return result;
   }
```

The second manipulates the strings as pointers and uses one loop to copy all the characters. The current pointer tracks the next location to write in the output string. The s1 and s2 pointers are used to travel down the input strings until we hit the terminating null characters.

```
   char *Weave2(char *s1, char *s2)
   {
      char *result, *cur;

      cur = result = (char *)GetBlock(strlen(s1) + strlen(s2) + 1);

      while (*s1 != '\0' || *s2 != '\0') { // one or both has chars
         if (*s1 != '\0')           // if chars left in s1 to copy,
            *cur++ = *s1++;         // copy s1 to cur, incr ptrs
         if (*s2 != '\0')           // do same for s2
            *cur++ = *s2++;
      }
      *cur = '\0';         // manually assign null char to end
      return result;       // return pointer to _head_ of string
}
```

Problem 2: Linked Lists

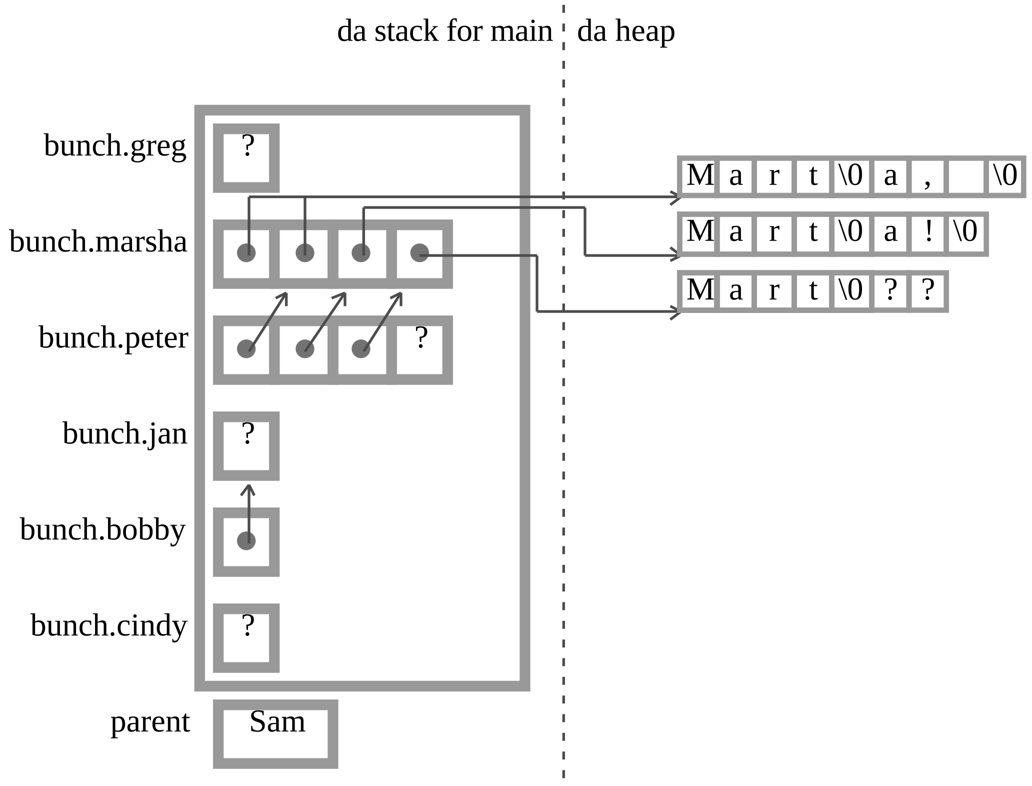
There are several tricky things to watch here, we need to keep a trailing pointer to the previous cell to split out an odd number, we have to handle the special case of deleting the first cell, and we have to take care to not access the contents of a cell after it has been freed. Whew!

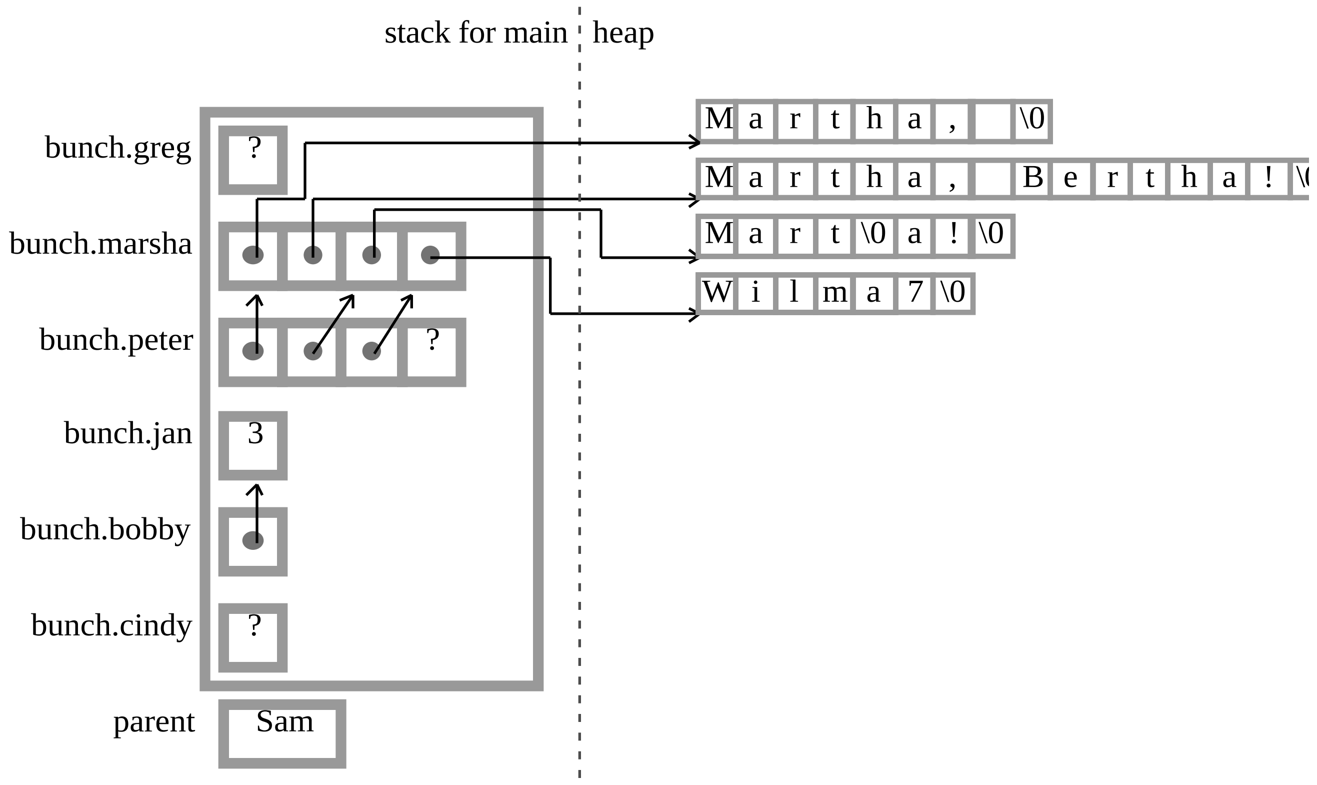
```
   static void RemoveOddNumbers(Cell **head)
   {
      Cell *prev = NULL, *cur, *next;

      for (cur = *head; cur != NULL; cur = next) {
         next = cur->next;
         if (cur->val % 2 == 1) {
            if (prev == NULL)
               *head = cur->next;
            else
               prev->next = cur->next;
            FreeBlock(cur);
         } else
            prev = cur;
      }
   }
```

For those of you already psyched about recursion and heading down that route, here is a recursive solution. The base case returns NULL when there are no cells left in the list, otherwise we handle the first cell and recur on the rest.

```
static void RemoveOdd(Cell **head)
{
   if (*head != NULL) {
      if ((*head)->val % 2 == 1) {
         Cell *toDelete = *head;
         *head = (*head)->next;
         FreeBlock(toDelete);
         RemoveOdd(head);
      } else
         RemoveOdd(&(*head)->next);
   }
}
```

Problem 3: Pointers and Program Tracing





Problem 4: Data structures

(4a) Write the HasIngredients function

```
static bool HasIngredients(Person *person, Recipe *recipe)
{
   int i;
   Ingredient needs, *has;

   for (i = 0; i < recipe->numIngreds; i++) {
      needs = recipe->ingreds[i];
      has = FindIngredient(person->pantry, person->numPantry, needs.name);
      if (!has || has->quantity < needs.quantity)
         return FALSE;
   }
   return TRUE;
}
```

(4b) Write the FindAssignment function

```
static Recipe *FindRecipeOfType(Person *person, DishType type)
{
   int i;
   Recipe *recipe;

   for (i = 0; i < person->numRecipes; i++) {
      recipe = person->recipes[i];
      if (recipe->type == type && HasIngredients(person, recipe))
         return recipe;
   }
   return NULL;
}
```

(4c) Write the AssignAll function

```
static bool ArrangePotluck(Potluck *dinner)
{
   DishType dish;
   int i;
   Person *person;
   Recipe *recipe = NULL;

   for (dish = Salad; dish <= Dessert; dish++) {
      for (i = 0; i < dinner->numAttending; i++) {
         person = &dinner->attendees[i];
         recipe = FindRecipeOfType(person, dish);
         if (recipe != NULL) {                     // found one
            dinner->menu[dish].chef = person;      // assign dish
            dinner->menu[dish].recipe = recipe;
            break;                              // no need to look further
         }
      }
      if (recipe == NULL) return FALSE;            // never found one
   }
   return TRUE;
}
```

Problem 5: File Processing

```
static Recipe *ReadOneRecipe(FILE *in)
{
    string name;
    Recipe *recipe;
    int i;

    name = ReadLine(in);
    if (name == NULL) return NULL;   // no more recipes to read
    recipe = New(Recipe *);
    recipe->name = name;
    fscanf(in, "Type: %d Num Ingredients: %d\n", &recipe->type,
                                          &recipe->numIngreds);
    recipe->ingreds = GetBlock(sizeof(Ingredient) * recipe->numIngreds);
    for (i = 0; i < recipe->numIngreds; i++) {
       recipe->ingreds[i].name = ReadLine(in);
       fscanf(in, "%d\n", &recipe->ingreds[i].quantity);
    }
    return recipe;
}
```