CS107 Handout 24S
Spring 2008 May 6, 2008

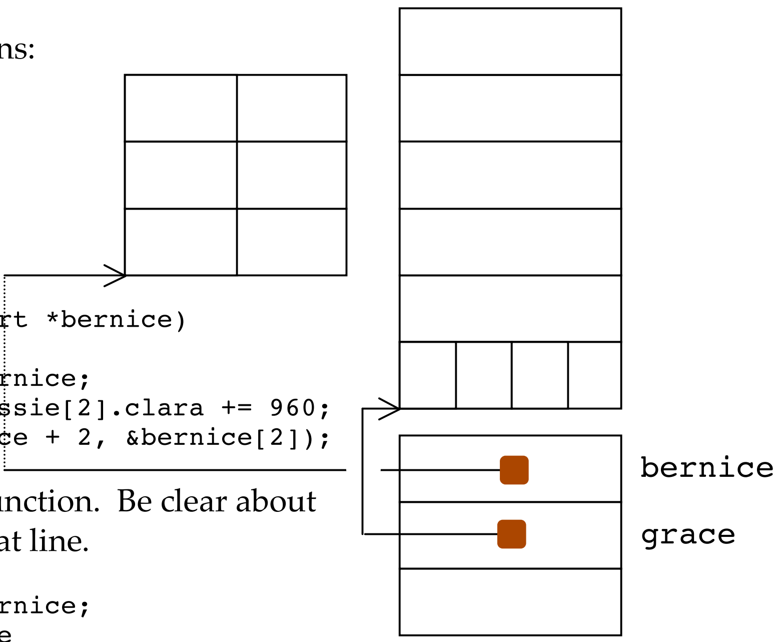
# Section Solution

## Solution 1: South Of Market

Consider the following **struct** definitions:

```
   typedef struct alley {
      char clementina[4];
      short **minna[3];
      struct alley *jessie;
      int clara;
   } alley;

   char *washburn(alley *grace, short *bernice)
   {
line 1    grace[2].clementina[12] = *bernice;
line 2    ((alley *)(grace->minna))->jessie[2].clara += 960;
line 3    return *(char **)washburn(grace + 2, &bernice[2]);
   }
```



Generate code for the entire **washburn** function. Be clear about what assembly code corresponds to what line.

```
// grace[2].clementina[12] = *bernice;
R1 = M[SP + 8];   // load bernice
R2 =.2 M[R1];     // load *bernice
R3 = M[SP + 4];   // load grace
M[R3 + 60] =.1 R2 // assign char at R3 + 2 * sizeof(alley) + 12 to be
*Bernice

// ((alley *)(grace->minna))->jessie[2].clara += 960;
R1 = M[SP + 4];   // load grace
R2 = R1 + 4;      // compute grace->minna (oh, that's an alley *)
R3 = M[R2 + 16];  // load jessie field of pretend struct
R4 = M[R3 + 68];  // load old value of int at R3 + 3 * sizeof(alley) –
sizeof(int)
R5 = R4 + 960;    // compute new value within register
M[R3 + 68] = R5;  // flush new value over old value

// return *(char **)washburn(grace + 2, &bernice[2]);
R1 = M[SP + 4];   // load grace
R2 = R1 + 48;     // compute grace + 2 * sizeof(alley)
R3 = M[SP + 8];   // load Bernice
R4 = R1 + 4;      // compute &bernice[2]
SP = SP – 8;      // make space for parameters
M[SP] = R2;
M[SP + 4] = R4;
CALL <washburn>;
SP = SP + 8;      // clean up params
RV = M[RV];       // update return value (note dereference)
RET;
```

## Solution 2: Matchmaking

```
vector generateAllCouples(vector *boys, vector *girls)
{
   vector couples;
   VectorNew(&couples, sizeof(couple), CoupleFree, 0);
   int i, j;
   couple item;

   for (int i = 0; i < VectorLength(boys); i++) {
      for (int j = 0; j < VectorLength(girls); j++) {
         item.boy = strdup(*(char **) VectorNth(boys, i));
         item.girl = strdup(*(char **) VectorNth(girls, j));
         VectorAppend(&couples, &item);
      }
   }

   return couples;
}
```

## Solution 3: `packPackets`

```
/**
 * Function: packPackets
 * ---------------------
 * Builds a contiguous array version of the packet list structure according
 * to the explanations provided on the prior page.  The parameter passed
 * is of type short *, because the first meaningful piece of information
 * stored in the list is a two-byte short.  The return value is of type
 * void *, because the implementation has no type information about the
 * packet data.
 */

void *packPackets(short *list)
{
   void *image = NULL;
   int imageSize = 0;
   while (list != NULL) {
      int packetSize = *list++;
      char *data = list;
      if (packetSize > 0) {
         image = realloc(image, imageSize + packetSize);
         memcpy((char *) image + imageSize, data, packetSize);
         imageSize += packetSize;
         list = (short *)(data + packetSize);
      } else {
         list = *(short **)data;   // list = *(void **)data would work too
      }
   }
   return image;
}
```