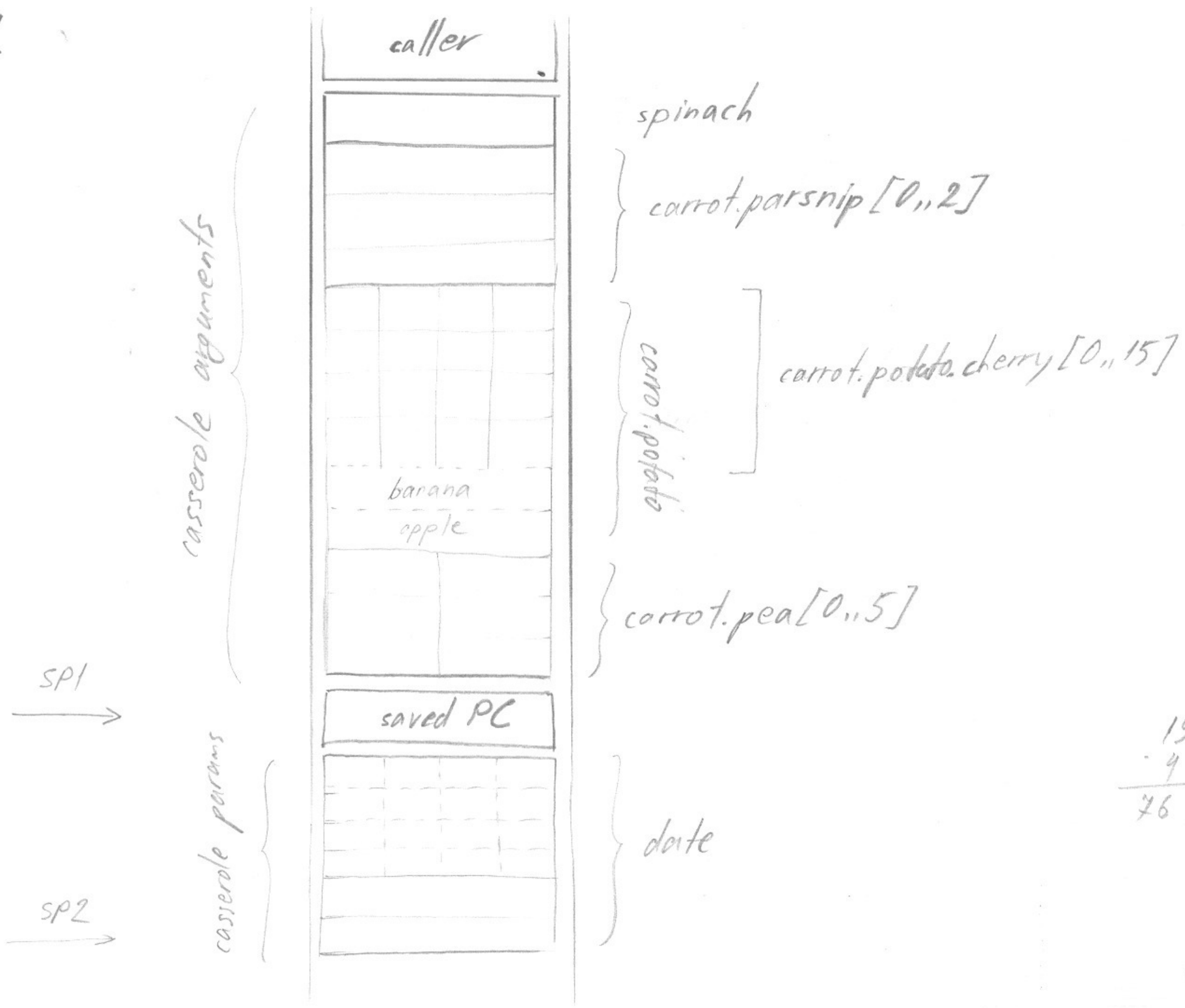
CS107 midterm

P1.



19
· 4
76

```
SP = SP + 24              // make room for date

// line 1
R1 = SP + 76              // & spinach
R2 = M[R1]                // spinach
R3 = SP + 40              // & carrot.potato.apple
R4 = M[R3]                // carrot.potato.apple integer
R4 = R4 * 2               // offset in shorts
R2 = R2 + R4              // & spinach → pea[carrot.potato.apple]
R8 = .2 M[R2]             // read its value as short
R1 = SP + 12              // & date.cherry[4]
M[R1] = .1 R8             // set date.cherry[4] char value.
```

16 · 4 = 64   19 · 4 = 76

```
// line 2
R1 = SP + 64          // & carrot.parsnip[0]
R1 = M[R1]            // carrot.parsnip[0]
R1 = R1 + 36          // (veggie*) • → parsnip
R1 = R1 + 16          // (veggie*) • → potato.banana
R2 = M[SP]            // load *(char**) & date
M[R1] = R2            // write that at R1

// line 3
R3 = SP + 4           // & date.banana
R3 = M[R3]            // date. banana
R3 = R3 + 4           // & date.banana[4]
R2 = .1 M[R3]         // load second tort parameter date.banana
                      // in R2

R3 = SP + 76          // & spinach
R3 = M[R3]            // spinach
R1 = R3 + 44          // &(spinach→parsnip[2]),
                      // which is tort first argument.

SP = SP + 8           // make room for tort args
M[SP] = R1            // load first param.
R3 = SP + 4
M[R3] = R2            // load second param.
CALL <tort>
SP = SP - 8           // come back up after tort is done

RV = RV + 240         // pointer arithmetic on fruits.

SP = SP - 24          // back up to saved PC

RET                   // out!
```

P2.

a.
```
void hashSetNew(" — ")
{
    hs → hashfn = hashfn;
    hs → freefn = freefn;
    hs → cmpfn = cmpfn;
    hs → elemSize = elemSize;
    hs → elems = malloc(64 *(elemSize + sizeof(bool)));
    hs → count = 0;                  should be made #define ...
    hs → allocLength = 64;
    for( int i = 0; i < allocLength; i ++)
    {
        * (bool*) ((char*) hs → elems + i*(hs → elemSize
            + sizeof(bool)) ) = false;
    } // bzeroing the whole thing would be more convenient,
      // but 'false' does not have a contract on being
      //                        language defined
      // necessarily zero.
}
```

b.
```
bool HashSetEnter( hashset* hs, void* elem)
{
    if (hs → count > (3 * hs → allocLength / 4))
        HashSetRehash (hs);
    int hash = hs → hashfn(elem, hs → allocLength);
    int i = 0;
    while( true )
    {
        hash =(hash + i) % hs → allocLength;
        i ++;
        void* place = (void*) ((char*) hs → elems + hash *
                            (hs → elemSize + sizeof(bool)) );
        if( *(bool*)place && ! hs → cmpfn ( elem, (bool*) place + 1)
            continue;
        // at this point we know that either element is
        // not present or it is present and is equal to elem.
        if(!*(bool*) place) hs → count ++; else {" — "}
            hs → freefn ( (bool*) place + 1);
        memcpy( (bool*) place + 1, elem, hs → elemSize);
        break;
    }
}
```

c.

```
static void HashSetRehash( hashset* hs)
{   int new_allocLength = 2 * hs→allocLength;
    void* new_array = malloc( new_allocLength
                      *( sizeof(bool) + hs→elemSize) );
   bzero( new_array,   new_allocLength * ( sizeof(bool) + hs→elemSize) );
   int old_allocLength = hs→allocLength;
   void* old_array     = hs→elems;
   hs→elems = new_array;
   hs→allocLength = new_allocLength;
   for( int i = 0; i < old_allocLength; i++)
   {
      void* place = (void*)((char*) old_array
                      + (sizeof(bool) + hs→elemSize) * i);
      if(!*(bool*)place) continue;
      HashSetEnter( hs, (bool*)place + 1 );
      // above line won't call rehash because enough space
      // is allocated.
   }
   free( old_array);   // our responsibility
}
```

— note that 'sizeof(bool) + hs→elemSize' should be made as a macro, but paper lacks 'editing' convenience to change it now.

P3.

```
person*  decompress (void*  image)
{
    int  n  =  *(int *) image;    // number of people
    void*  start  =  (void*)((int*) image + 1);
    person*  persons  =  (person *) malloc( n * sizeof(person) );

    for( int  i=0;  i < n;   i++)
    {
        person*  p  =  persons + i;
        p → name  =  strdup(start);
        int forward  =  strlen(start)+1;
        if(forward % 4  != 0)   // fill padding
            forward = forward + (4-forward % 4);
        // move start forward:
        start = (void*)((char*) start + forward);
        p → numFriends  = *(int *) start;
        start = (void *)( (int*) start + 1);
        p → friends  =  (char**) malloc(p→numFriends * sizeof(char*))
        for ( int j = 0;  j < p → numFriends;  j++)
        {   *(p → friends + j)  =  strdup( start );
            start  =  (void *)( (char **) start + 1);
        }
    }
    return  persons;
}
```