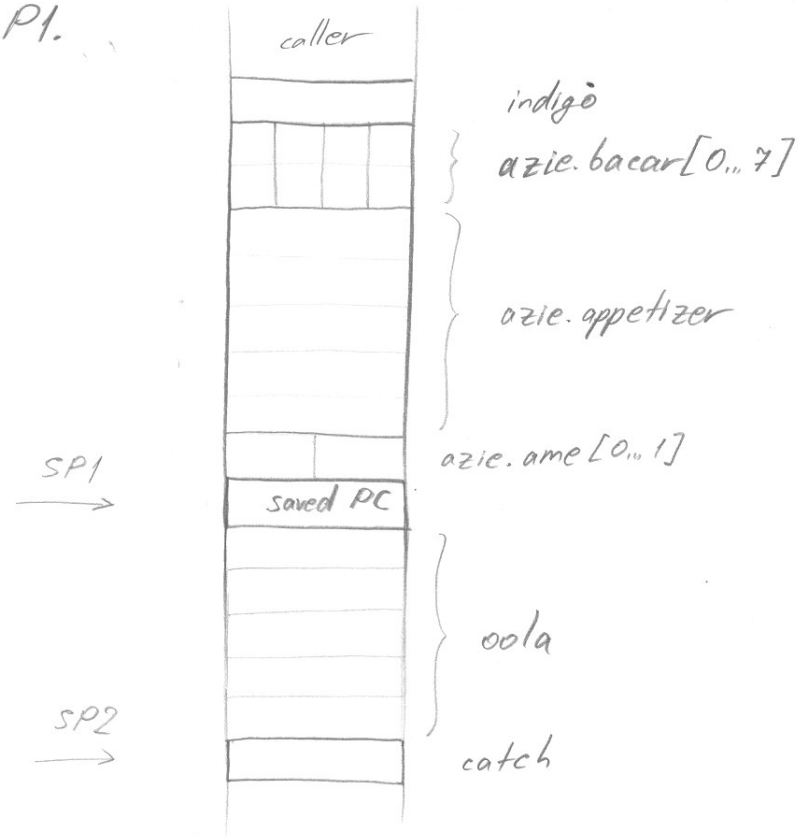
CS 107 midterm practice

P1.




```
SP = SP - 20          // space for oola
SP = SP - 4           // space for catch

// line 1
R1 = M[SP]            // value of catch
R1 = R1 + 4           // & catch → farallon
R1 = R1 + 4           // & catch → farallon.aqua[0]
R2 = 4 * 4            // int array offset
R1 = R1 + R2          // & catch → farallon.aqua[4]
R8 = M[R1]            // load catch → farallon.aqua[4]

R1 = SP + 24          // & saved PC
R1 = R1 + 4           // & azie (same as & azie.ame[0])
R2 = 2 * 2            // short array offset
R1 = R1 + R2          // & azie.ame[2]
R9 = .2 M[R1]         // azie.ame[2]
```

```
R1 = SP + 24
R1 = R1 + 4
R1 = R1 + 24          // & azie.bacar[0]
R2 = R9 * 1           // offset for clar array
R1 = R1 + R2          // & azie.bacar[azie.ame[2]]
R10 = .1 M[R1]        // load char at base addr R1
R10 = R10 + R8        // do the addition
M[R1] = R10           // write result back at addr R1.
```

R1 = SP + 40

```
// line 2
R1 = SP + 4           // & oola
R1 = R1 + 16          // & oola.quince
R1 = R1 + 4           // imagin R1 was address to
                      // some desert and jump to its
                      // farallon (same as &_.farallon.garydank
R1 = R1 + 16          // same logic, all the way
M[R1] = 0             // write 0 to that addr.
```

```
// line 3
// load second arg in R8
R8 = SP + 4           // & oola
// load first arg in R9
R1 = SP + 24          // & saved PC
R1 = R1 + 36          // & indigo
R9 = R1               // load & indigo in R9

SP = SP - 8           // make space for args to call
                      // function dinnerisserved
M[SP] = R8            // load second arg
R1 = SP + 4           // load first arg
M[R1] = R9
```

```
CALL <dinnerisserved>
SP = SP + 8                 // get back up
// at this point, RV holds  dessert *
R1 = M[RV]                  // dessert base address
R1 = R1 + 4                 // & ␣.farallon
R1 = R1 + 4                 // & ␣.farallon.aqua[0]
RV = R1                     // load that addr to return
SP = SP + 24                // back up to & saved PC
RET
```

P2.

```
void* packetize( const void* image,
                 int size,
                 int packetSize)
{
    void*  ret;
    void** prevPtr = &ret;
    char* src = (char*) image;
    for( int i = 0; i <= fullPacketCnt; i++)
    {
        int allocSize = (i == fullPacketCnt) ? remainderSize : packetSize;
        void* dest = malloc(allocSize + sizeof(void*));
        memcpy(dest, src, allocSize);
        src = src + allocSize;   // update source for next iteration.
        memcpy( prevPtr, &dest, sizeof(void*) );                    (&)
        prevPtr = (void**)((char*) dest + allocSize);
        if (i == fullPacketCnt)
            bzero(prevPtr, sizeof(void*));
    }
return ret; }
```

P3.

(a)

```
void MultiSetNew("---")
{
    ms->elemSize = elemSize;
    ms->free     = free;   // do we need to store this?
    HashSetNew(&(ms->elements),
               elemSize + sizeof(int),
               numBuckets,
               hash, compare, free);
}

void MultiSetDispose("---")
{
    HashSetDispose(&(ms->elements));
}
```

(b)

```
void MultiSetEnter("---")
{
    void* temp = malloc(ms->elemSize + sizeof(int));
    memcpy(temp, elem, elemSize);
    int one = 1;
    memcpy((char*)temp + elemSize, &one, sizeof(int));
    if (HashSetLookup(&(ms->elements), temp) == NULL)
    {
        HashSetEnter(&(ms->elements), temp);
    } else {
        // knowing hashset does not know about integer
        // at the end, we can mess with it without
        // breaking hashset:
        void* found = HashSetLookup("---");
        int cnt = *(int*)((char*)found + ms->elemSize);
        cnt++;
        memcpy((char*)found + ms->elemSize, &cnt, sizeof(int));
    }
    free(temp);
}
```

(c)

```
typedef struct
{
    int elemSize;
    MultiSetMapFunction mapfn;
    void * auxData;
} helper;

void connectorfn( void * elemAddr, void * auxData)
{
    helper * data = (helper*) auxData;
    int count = *(int*)((char*)elemAddr + data->elemSize);
    MultiSetMapFunction mapfn = data->mapfn;
    void * auxData = data->auxData;
    mapfn(elemAddr, count, auxData);
}

void MultiSetMap(" ")
{
    helper helper;
    helper.auxData = auxData;
    helper.elemSize = ms->elemSize;
    helper.mapfn = map;
    HashSetMap(&(ms->elements), connectorfn, &helper);
}
```

P4.

This is a simple application of P3's MultiHashSet.

```
void mapfn(void* elem, int cnt, void* auxData)
{
    maxTicketsStruct* mts = (maxTicketsStruct*) auxData;
    if (mts->numTickets < cnt)
    {
        mts->licencePlate = ~~*(char**)elem;~~ (char*) elem;
        mts->numTickets   = cnt;
    }
}

void FindQueenOfParkingInfractions("_____")
{
    maxTicketsStruct auxData;
    auxData.numTickets = -1;
    auxData.licencePlate = NULL;
    MultiSetMap(ms, mapfn, &auxData);
    strcpy(licencePlateQueen, auxData.licencePlate);
}
```

P5.

(a) Because they don't fit in a single instruction?

(b) Saved PC cannot be above arguments by the same reason as to why arguments are read right → left.

It cannot be below callee's local parameters, because caller does not know how callee is implemented.

(c)
```
bool IsLittleEndian() {
    int* four;
    *four = 1;
    short* two = (short*)((char*) four + sizeof(short));
    return *two == 0; }
```