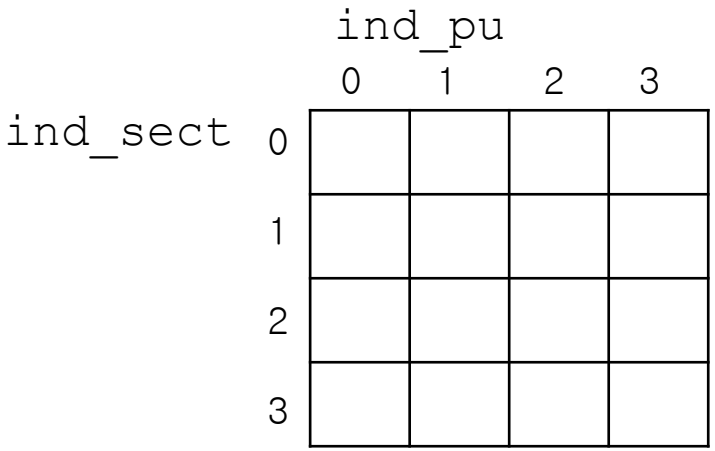
Ex4. Flash read verification

1. Formal verification of a flash memory reading unit
   - Show the correctness of the `flash_read()`
     - By using randomized testing
       - Randomly select the physical sectors to write four characters and set the corresponding SAMs
     - By using exhaustive testing
       - Create 43680 (16*15*14*13) distinct test cases
         - » Do not print test cases in your hardcopy to save trees
     - By using CBMC
       - Create environment model satisfying the invariant formula by using `__CPROVER_assume()` and nested loops

```
typedef struct _SAM_type{
    unsigned char offset[SECT_PER_U];
}SAM_type;
typedef struct _PU_type{
    unsigned char sect[SECT_PER_U];
}PU_type;

// Environment assumption
// 0. Each unit contains 4 sectors.
// 1. There is one logical unit containing "abcd"
// 2. There are 4 physical units
// 3. The value of SAM table is 255 if the corresponding
//    physical sector does not have a valid data
void flash_read(char *buf, SAM_type *SAM, PU_type *pu ){
    unsigned char nSamIdx = 0;
    unsigned char pu_id = 0;
    unsigned char n_scts = 4; // number of sectors to read
    unsigned char offset = 0; //offset of the physical sector to read
    unsigned char pBuf = 0;

    while(n_scts > 0){
          pu_id=0;
          offset = 255;
          // read 1 character
          while(1) {
                    if (SAM[pu_id].offset[nSamIdx] != 255){
                              offset = SAM[pu_id].offset[nSamIdx++];
                              buf[pBuf] = PU[pu_id].sect[offset];
                              break;
                    }
                    pu_id ++;
          }
          n_scts--;
          pBuf ++;
    }
}
```

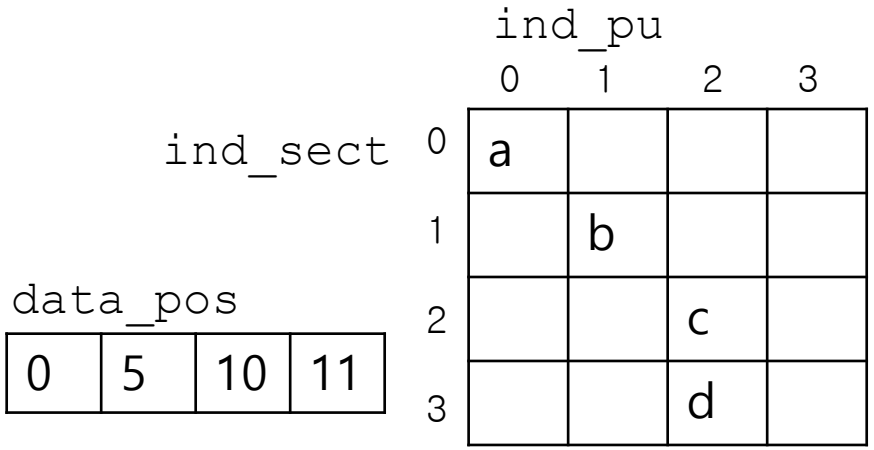
• Problem #1. Random solution

```
void main(){
char res[50];
int tc = 0;
int count = 0;
int nTC = 43680;// # of possible distribution
                // 16*15*14*13
while(tc++ < nTC){
     randomized_test();
     flash_read(&res[count],SAM,pu);
     assert(res[0] == 'a' && res[1] == 'b' &&
            res[2] == 'c' && res[3] =='d');
}}
```




```
 #include <stdio.h>
#include <time.h>
#include <assert.h>
#define SECT_PER_U 4
#define NUM_PHI_U 4

typedef struct _SAM_type{
    unsigned char offset[SECT_PER_U];
SAM_type;

typedef struct _PU_type{
   unsigned char sect[SECT_PER_U];
}PU_type;

char data[SECT_PER_U] = "abcd";

PU_type pu[NUM_PHI_U];
SAM_type SAM[NUM_PHI_U];

void randomized_test(){
    unsigned int i = 0, j = 0;
    unsigned char ind_pu, ind_Sect;
    // Initialization
    for(i = 0;i < NUM_PHI_U;i++){
        for(j = 0;j < SECT_PER_U;j++){
            SAM[i].offset[j] = 255;
            pu[i].sect[j] = 0;
    }}

    while (i< SECT_PER_U) {
       ind_pu  = rand()%4;
       ind_Sect= rand()%4;
       if(pu[ind_pu].sect[ind_Sect] == 0){
          pu[ind_pu].sect[ind_Sect] = data[i];
          SAM[ind_pu].offset[i] = ind_Sect;
          i++;
       }
    }
 }
```

- Problem #1.
  Exhaustive solution




```
void exhaustive_test(int *data_pos){
    unsigned int i = 0, j = 0;
    unsigned char ind_pu, ind_Sect;

    for(i = 0;i < NUM_PHI_U;i++){
        for(j = 0;j < SECT_PER_U;j++){
            SAM[i].offset[j] = 255;
            pu[i].sect[j] = 0;
        }
    }

    for(i = 0;i < NUM_PHI_U;i++){
        ind_pu = data_pos[i]/4;
        ind_Sect = data_pos[i]%4;
        pu[ind_pu].sect[ind_Sect] = data[i];
        SAM[ind_pu].offset[i] = ind_Sect;
    }
}
```

```
void main(){
char res[4];
int i, j,k,l, data_pos[4];
 //# of all distributions = 16*15*14*13
for(i = 0;i < NUM_PHI_U * SECT_PER_U;i++){
    for(j = 0;j < NUM_PHI_U * SECT_PER_U;j++){
        if (j == i) continue;
        for(k = 0;k < NUM_PHI_U * SECT_PER_U;k++){
            if (k == i || k == j) continue;
            for(l = 0;l < NUM_PHI_U * SECT_PER_U;l++){
                if(l == i || l == j || l == k) continue;

                data_pos[0] = i;
                data_pos[1] = j;
                data_pos[2] = k;
                data_pos[3] = l;
                exhaustive_test(data_pos);
                flash_read(&res[count],SAM,pu);
                assert(res[0] == 'a' && res[1] == 'b'
                && res[2] == 'c' && res[3] == 'd');
} } } } }
```

- Problem #1. CBMC solution

```
void CBMC_environ_setting(){
    unsigned int i = 0, j = 0;
    unsigned char ind_pu, ind_Sect;
    for(i = 0;i < NUM_PHI_U;i++){
        for(j = 0;j < SECT_PER_U;j++){
            SAM[i].offset[j] = 255;
            pu[i].sect[j] = 0;
    }}

    for(i = 0;i < SECT_PER_U;i++){
        ind_pu = nondet_char();
        ind_Sect = nondet_char();
        __CPROVER_assume(ind_pu>=0 && ind_pu <NUM_PHI_U);
        __CPROVER_assume(ind_Sect>=0 && ind_Sect <SECT_PER_U);
        __CPROVER_assume(pu[ind_pu].sect[ind_Sect] == 0);

        pu[ind_pu].sect[ind_Sect] = data[i];
        SAM[ind_pu].offset[i] = ind_Sect;
    }
}
void main(){
  char res[50];
  int count = 0;
  CBMC_environ_setting();
  flash_read(&res[count],SAM,pu);
  assert(res[0] == 'a' && res[1] == 'b'  && res[2] == 'c' && res[3] == 'd');}
```