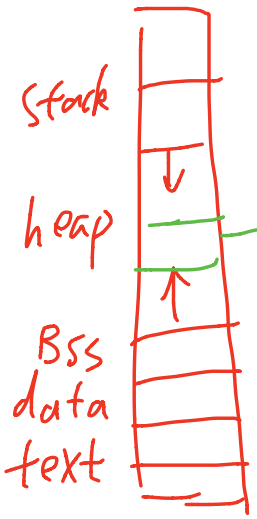
Dynamic Memory allocation

```
// dynamic memory allocation in processes using malloc()

#include <stdio.h>
#include <stdlib.h>

main() {

int x;
int *y;

  x = 10;

  y = (int *) malloc(sizeof(int));

  *y = 20;

  fprintf(stdout,"%d %d\n",x,*y);

}
```

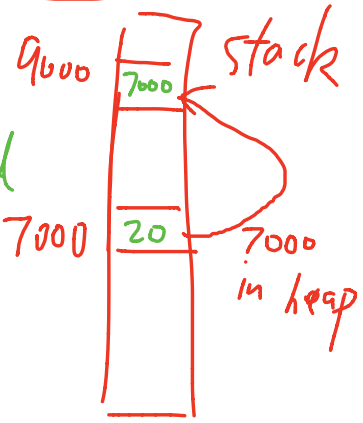
this may change depends on the architecture or system

4 byte

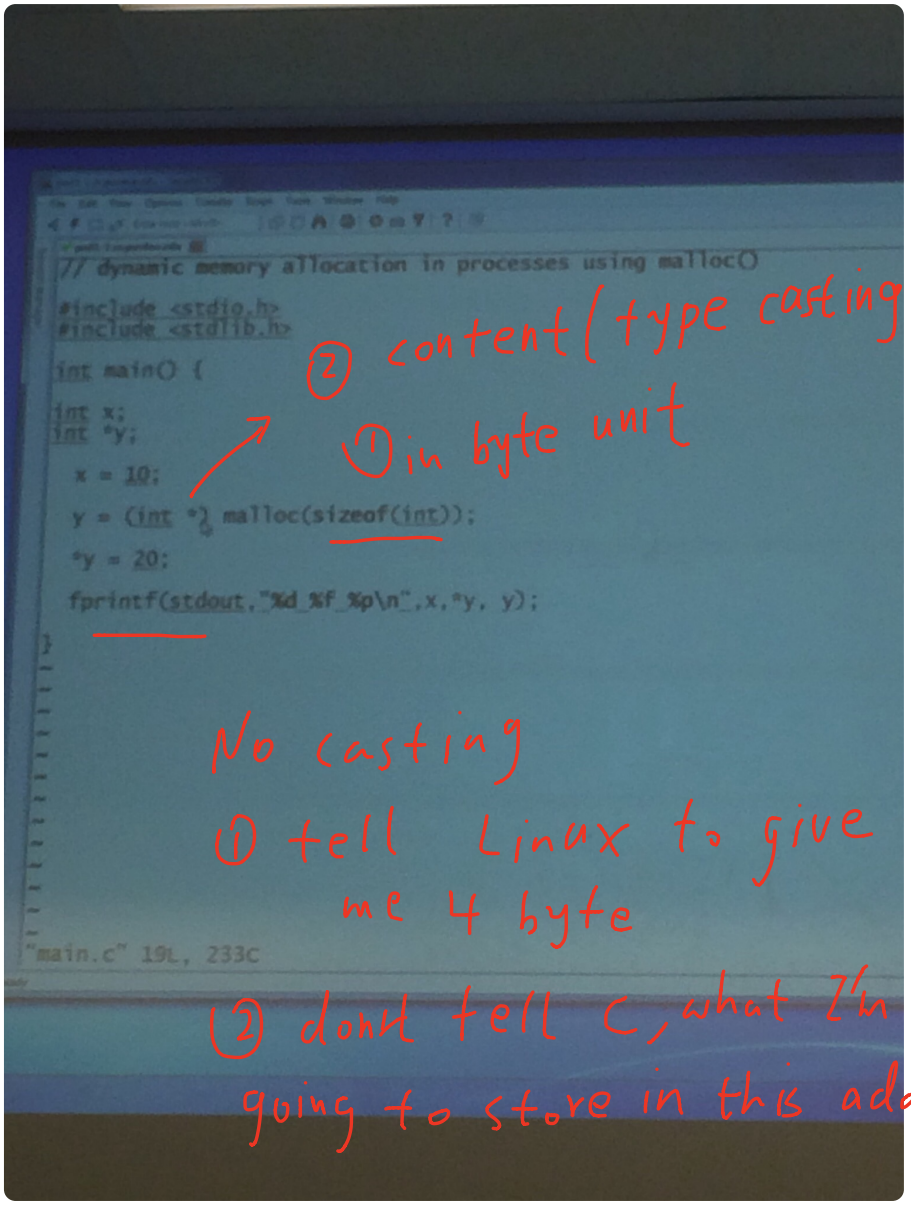
check the return value of malloc, since it may fail.

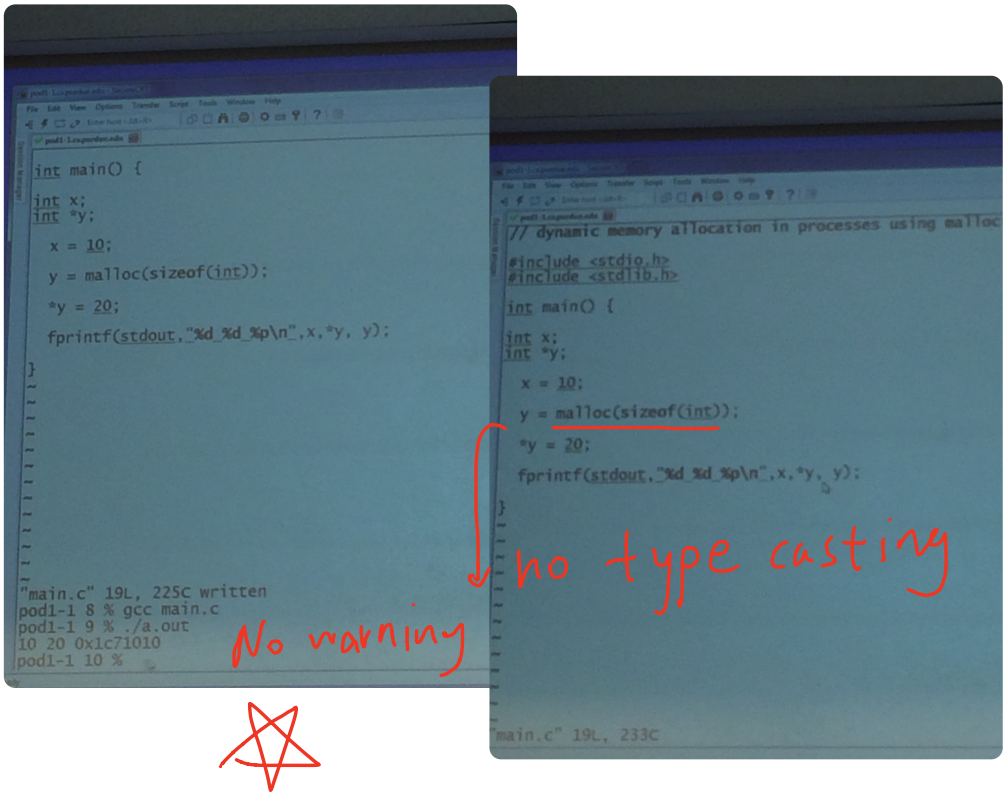


4 MB contiguously located or fail

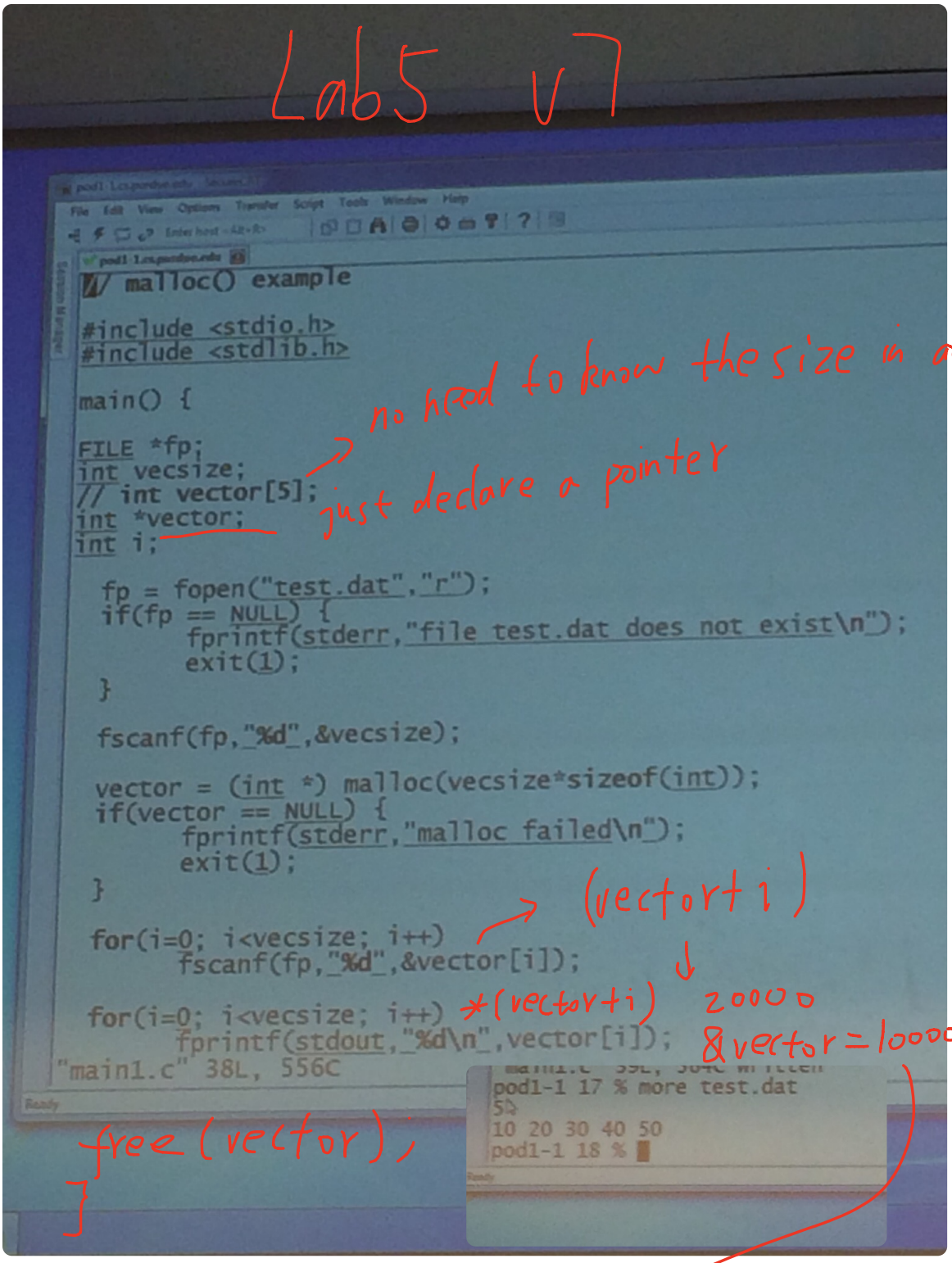


malloc() vs brk() system call


// dynamic memory allocation in processes using malloc()
#include <stdio.h>
#include <stdlib.h>
int main() {
int x;
int *y;
x = 10;
y = (int *) malloc(sizeof(int));
*y = 20;
fprintf(stdout,"%d %f %p\n",x,*y, y);
}
"main.c" 19L, 233C
② content (type casting)
① in byte unit
No casting
① tell Linux to give me 4 byte
② don't tell C, what I'm going to store in this address


int main() {
int x;
int *y;
x = 10;
y = malloc(sizeof(int));
*y = 20;
fprintf(stdout,"%d %d %p\n",x,*y, y);
}
"main.c" 19L, 225C written
pod1-1 8 % gcc main.c
pod1-1 9 % ./a.out
10 20 0x1c71010
pod1-1 10 %
No warning
// dynamic memory allocation in processes using malloc
#include <stdio.h>
#include <stdlib.h>
int main() {
int x;
int *y;
x = 10;
y = malloc(sizeof(int));
*y = 20;
fprintf(stdout,"%d %d %p\n",x,*y, y);
}
"main.c" 19L, 233C
no type casting

Lab5 v7



```
// malloc() example

#include <stdio.h>
#include <stdlib.h>

main() {

FILE *fp;
int vecsize;
// int vector[5];
int *vector;
int i;

  fp = fopen("test.dat","r");
  if(fp == NULL) {
        fprintf(stderr,"file test.dat does not exist\n");
        exit(1);
  }

  fscanf(fp,"%d",&vecsize);

  vector = (int *) malloc(vecsize*sizeof(int));
  if(vector == NULL) {
        fprintf(stderr,"malloc failed\n");
        exit(1);
  }

  for(i=0; i<vecsize; i++)
        fscanf(fp,"%d",&vector[i]);

  for(i=0; i<vecsize; i++)
        fprintf(stdout,"%d\n",vector[i]);
"main1.c" 38L, 556C
```

```
pod1-1 17 % more test.dat
5
10 20 30 40 50
pod1-1 18 %
```

no need to know the size in advan

just declare a pointer

(vector+i)

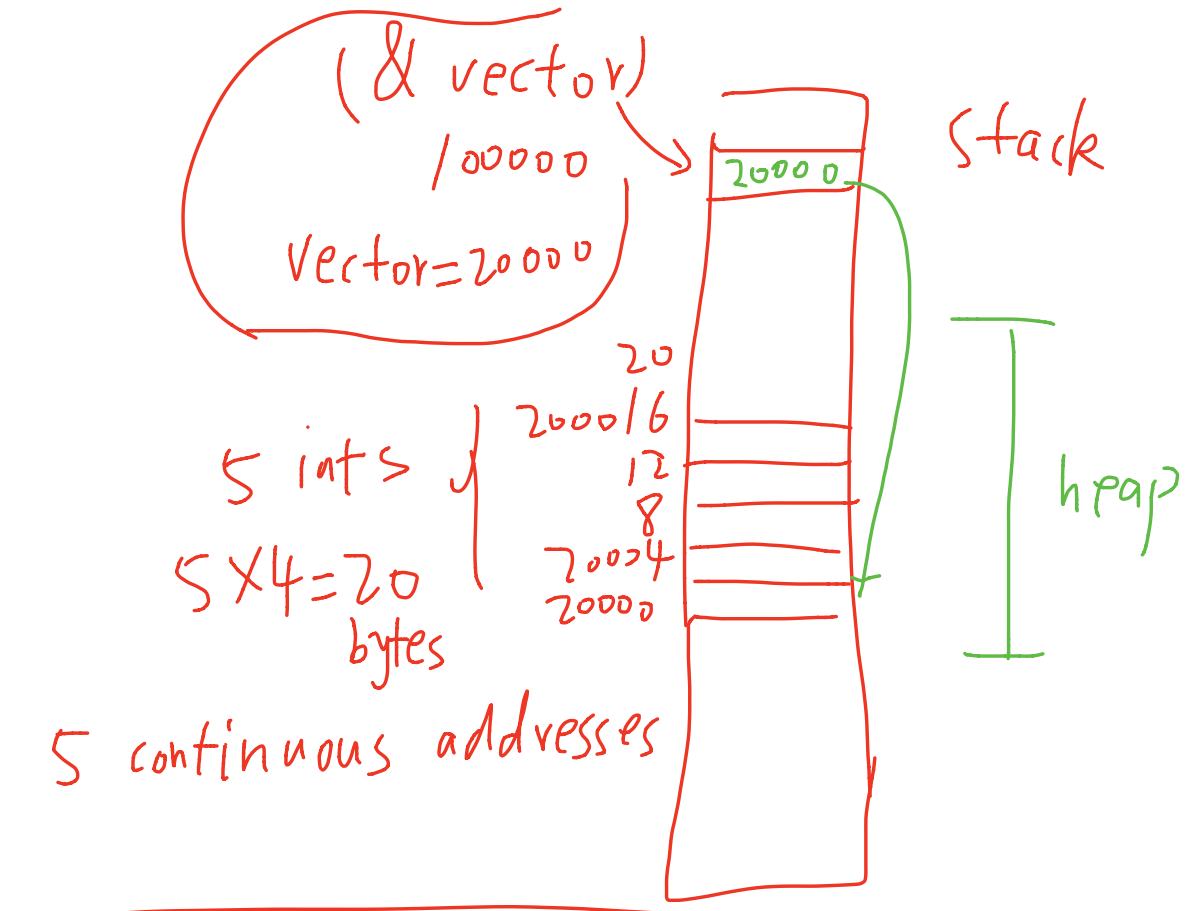
*(vector+i)

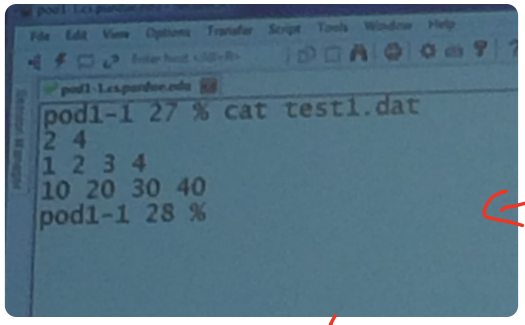
20000

&vector=100000

free(vector);

}





main1a.c

data file

```c
// malloc() example involving 2-D tables

#include <stdio.h>
#include <stdlib.h>

int main() {

FILE *fp;
int mx, my;
int **twod;
int i, j;

  fp = fopen("test1.dat","r");
  if(fp == NULL) {
        fprintf(stderr,"file test.dat does not exist\n");
        exit(1);
  }

  // read the size of 2-d table
  fscanf(fp, "%d %d", &mx, &my);

  // allocate space for pointers for mx rows
  twod = (int **) malloc(mx * sizeof(int *));
  if(twod == NULL) {
        fprintf(stderr,"malloc failed\n");
        exit(1);
  }

  // for each row, allocate space for my integers
  for(i=0; i<mx; i++)
        *(twod + i) = (int *) malloc(my * sizeof(int));

"main1a.c" 41L, 767C
```

mx=2 my=4 in test1.dat

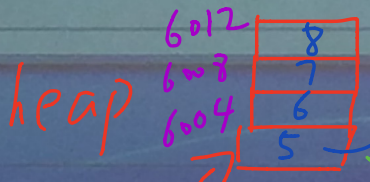
Not hard code

4 → 32 bit

8 → 64 bit

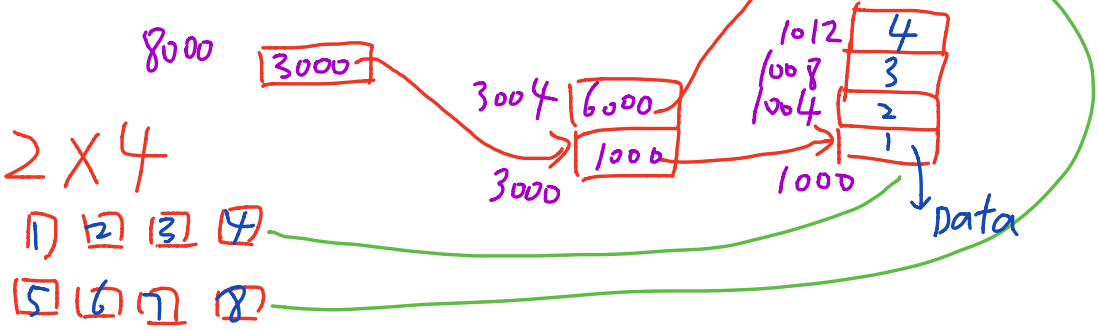
(not 8 Byte)

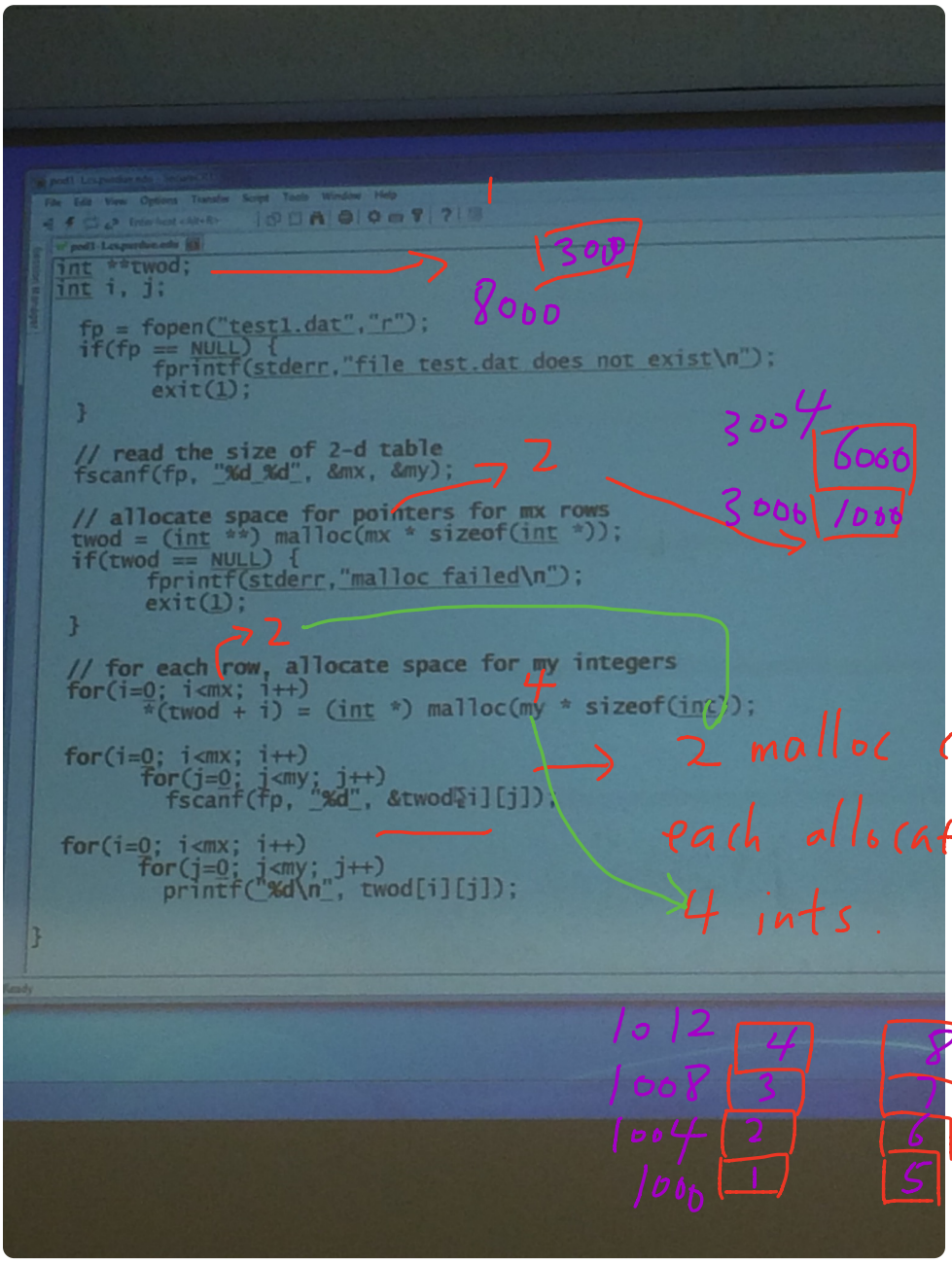
not depend on Architecture use sizeof( )



heap

stack




int **twod;
int i, j;
fp = fopen("test1.dat","r");
if(fp == NULL) {
fprintf(stderr,"file test.dat does not exist\n");
exit(1);
}
// read the size of 2-d table
fscanf(fp, "%d %d", &mx, &my);
// allocate space for pointers for mx rows
twod = (int **) malloc(mx * sizeof(int *));
if(twod == NULL) {
fprintf(stderr,"malloc failed\n");
exit(1);
}
// for each row, allocate space for my integers
for(i=0; i<mx; i++)
*(twod + i) = (int *) malloc(my * sizeof(int));
for(i=0; i<mx; i++)
for(j=0; j<my; j++)
fscanf(fp, "%d", &twod[i][j]);
for(i=0; i<mx; i++)
for(j=0; j<my; j++)
printf("%d\n", twod[i][j]);
}
1
3000
8000
2
3004
6000
3000
1000
2
4
2 malloc call,
each allocate 4 units
4 ints.
1012
1008
1004
1000
4
3
2
1
8
7
6
5
6012
6008
6004
6000